Universität der Bundeswehr
Fakultät für Informatik – Institut 2
Prof. Dr. Wolfgang Hommel
Michael Grabatin, Michael Steinke

WT 2020
Übungsblatt 2

Einführung in die Informatik II

24. und 27.01.2020

1 Permutationen – Ordnung

Wir bezeichnen eine einstufige Reihung P mit n Elementen als *Permutation*, wenn jeder der Indizes $0, 1, ..., n-1$ von P genau einmal als Wert in P vorkommt.

a) Vervollständigen Sie die folgende Funktion, die mit Hilfe einer charakteristischen Funktion cfm prüft, ob der Parameter P in diesem Sinn eine Permutation ist.

```scala
def isPerm(p : Array[Int]) : Boolean = {
  val n = p.length
  var cfm = Array.fill(n)(false)
  //...
}
```

b) Geben Sie die lexikographisch größte Permutation der Zahlen $0, 1, ..., 8$ an.

c) Erläutern Sie, warum die Permutationen

$$\left|\begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 0 & 3 & 5 & 1 & 8 & 4 & 7 & 6 & 2 \end{array}\right| \text{und} \left|\begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 0 & 3 & 5 & 1 & 8 & 6 & 2 & 4 & 7 \end{array}\right|$$

lexikographisch unmittelbar direkt aufeinander folgen.

d) Geben Sie zu

$$\left|\begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 0 & 3 & 2 & 1 & 5 & 8 & 7 & 6 & 4 \end{array}\right|$$

die lexikographisch nächste Permutation an.

e) Entwickeln Sie ein Scala-Programmstück, welches zu einer im Array P gegebenen Permutation im Array Q die nächstgrößere Permutation herstellt.

f) Testen Sie das Programmstück ebenso wie das Ergebnis von Teilaufgabe a mit geeigneten Daten am eigenen Rechner.

2 Permutation – Zyklus

Sei P eine Permutation der Länge n und sei i einer der Indizes $0, 1, ..., n-1$ von P. Durch wiederholte Anwendung von P auf i erhält man den *Zyklus* von i in P.

Beispiel: In der Permutation aus Aufgabe 1d findet man zu 4 den Zyklus $(4, 5, 8)$, weil P 4 auf 5, 5 auf 8 und 8 auf 4 abbildet.

$$\begin{vmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 0 & 3 & 2 & 1 & 5 & 8 & 7 & 6 & 4 \end{vmatrix}$$

a) Geben Sie zu allen Permutationen aus der Angabe von Aufgabe 1 jeweils alle verschiedenen Zyklen an (der Zyklus von 8 ist offenbar gleich dem Zyklus von 4 bzw. von 5).

b) Gegeben seien die Typvereinbarungen

```scala
type Perm = Array[Int]
type Cycles = List[List[Int]]
```

Dann berechnet die folgende Funktion zu einer gegebenen Permutation alle seine Zyklen:

```scala
def cyclesOf(p : Perm) : Cycles = {
  val n = p.length
  var cs : Cycles = List()
  var cfm = Array.fill(n)(false)
  for (i <- 0 to n - 1) {
    if (!cfm(i)) {
      cfm(i) = true
      var cycle = List(i)
      var j = p(i)
      while (j != i) {
        cfm(j) = true
        cycle = cycle ::: List(j)
        j = p(j)
      }
      cs = cs ::: List(cycle)
    }
  }
  return cs
}
```

Die folgende Prozedur gibt die Zyklen einer Permutation aus:

```scala
def printCycles(p : Perm) : Unit = {
  for (c <- cyclesOf(p)) println(c.mkString("(", ", ", ")"))
}
```

Was gibt `printCycles(P)` für alle Permutationen aus der Angabe von Aufgabe 1 aus?

c) Aus den mit `cyclesOf(P)` berechneten Zyklen lässt sich die Permutation P zurückgewinnen. Vervollständigen Sie dazu folgende Scala-Funktion:

```scala
def permByCycles(cs : Cycles) : Perm = {
  //groessten enthaltenen Zahlenwert in cs bestimmen
  def maxEl(cs: Cycles): Int = {
  ...
  }
  var p = Array.fill(maxEl(cs) + 1)(0)
  ...
  return p
}
```

3 Damenproblem

Die folgenden gegebenen Permutationen stellen Lösungen des Damenproblems, wie in der Vorlesung (Kap. 1, Folie 54) definiert, dar.

```
val P = Array(7, 3, 0, 2, 5, 1, 6, 4)
val Q = Array(5, 2, 4, 6, 0, 3, 1, 7)
```

a) Die Funktion `spiegle` spiegelt eine beliebige Permutation an der Hauptdiagonalen. Wir betrachten Aufrufe der folgenden **fehlerhaften** Implementierung:

```
def spiegleF(p: Perm): Perm = {
  var res = Array.fill(8)(0)
  for (i <- 0 to 7) res(P(i)) = i
  return res
}
```

Der Aufruf `spiegleF(P)` ergibt die (korrekte) Antwort `Array(2, 5, 3, 1, 7, 4, 6, 0)`. Das gleiche Ergebnis liefern aber auch die Aufrufe `spiegleF(spiegleF(P))` und `spiegleF(Q)`. Warum?

b) Mit folgendem beispielhaften Aufruf wollen wir feststellen, ob die Permutation P lexikographisch kleiner ist als die Permutation Q.

```
lexKleiner(P, Q)
```

Ergänzen Sie die folgende Definition entsprechend!

```
def lexKleiner(p : Perm, q : Perm) : Boolean = ...
```

c) Eine Lösung P des Damenpromblems gelte als „neu“, wenn sie sich **nicht** durch Spiegelungen und/oder Drehungen in eine lexikographisch kleinere Permutation transformieren lässt. Vervollständigen Sie in diesem Sinn die folgende Definition!

```
def drehe(p: Perm): Perm = {
    var Erg = Array.fill(8)(0)
    for(i <- (0 to 7)) Erg(p(i)) = p.length - 1 - i
    return Erg
}

def istNeu(p : Perm) : Boolean = ...
```

Hinweis: Aus der Vorlesung bekannte Prozeduren und Funktionen können verwendet werden.