P03 – Subsetting und Funktionen

26. April 2021

Contents

Hinweise zur Abgabe:

Erstelle pro Aufgabe eine R-Code-Datei und benenne diese nach dem Schema `P<Woche>-<Aufgabe>.R` also hier `P03-1.R`, `P03-2.R` und `P03-3.R` Schreibe den Code zur Lösung einer Aufgabe in die jeweilige Datei.

Es ist erlaubt (aber nicht verpflichtend) zu zweit abzugeben. Abgaben in Gruppen von drei oder mehr Personen sind nicht erlaubt. Diese Gruppierung gilt nur für die Abgabe der Programmierprobleme, nicht für die Live-Übungen.

Bei Abgaben zu zweit gibt nur eine der beiden Person ab. Dabei müssen in **jeder** abgegebenen Datei in der **ersten Zeile** als Kommentar **beide** Namen stehen also zB

```
# Ada Lovelace, Charles Babbage

1+1
# ...
```

Die Abgabe der einzelnen Dateien (kein Archiv wie `.zip`) erfolgt über Moodle im Element namens `P03`. Die Abgabe muss bis spätestens Sonntag, 2. Mai 2021, 23:59 erfolgen.

1 Verknüpfung von Funktionen (5 Punkte)

Erstelle einen Infix-Operator `%o%`, der Funktionen verknüpft, sodass mit `fg <- f %o% g` die Ausgabe von `fg(x)` gleich der Ausgabe von `f(g(x))` ist. Wir nehmen an, dass das linke und rechte Argument jeweils Funktionsobjekte sind. Außerdem operiert die linke Funktion auf genau einem Argument, wohingegen keine Vorgaben bzgl der Anzahl der Argumente für die rechte Funktion gemacht werden sollen.

Hinweis: ...

```
mean(1,2,3,4)
## [1] 1
mean_c <- mean %o% c
mean_c(1,2,3,4)
## [1] 2.5
```

2 Nebendiagonale (15 Punkte)

1. Die Funktion `diag()` gibt die (Haupt-)Diagonale einer Matrix aus. Schreibe eine Funktion `n_diag()` mit zwei Argumenten `mat` und `n`, die die Nebendiagonalen der Matrix `mat` als Vektor ausgibt. Für

positive `n` soll die `n`-te Nebendiagonale oberhalb der Hauptdiagonale ausgegeben werden, für negative die `abs(n)`-te unterhalb. Wir nehmen an, dass `mat` eine quadratische Matrix ist.

```
(x <- matrix(1:16, nrow=4))
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
n_diag(x, -3)
## [1] 4
n_diag(x, -2)
## [1] 3 8
n_diag(x, -1)
## [1]  2  7 12
n_diag(x, 0)
## [1]  1  6 11 16
n_diag(x, 1)
## [1]  5 10 15
n_diag(x, 2)
## [1]  9 14
n_diag(x, 3)
## [1] 13
```

2. Mit `diag(mat) <- x` wird die Diagonale der Matrix `mat` auf `x` gesetzt. Schreibe die zu `n_diag()` gehörende Ersetzungsfunktion.

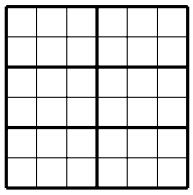
```
(x <- matrix(1:9, nrow=3))
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
n_diag(x, -2) <- -1
x
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]   -1    6    9
n_diag(x, -1) <- -2
x
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]   -2    5    8
## [3,]   -1   -2    9
n_diag(x, 0) <- -1:-3
x
##      [,1] [,2] [,3]
## [1,]   -1    4    7
## [2,]   -2   -2    8
## [3,]   -1   -2   -3
n_diag(x, 1) <- -11:-12
x
##      [,1] [,2] [,3]
## [1,]   -1  -11    7
## [2,]   -2   -2  -12
```

```
## [3,]   -1   -2   -3
n_diag(x, 2) <- -42
x
##      [,1] [,2] [,3]
## [1,]   -1  -11  -42
## [2,]   -2   -2  -12
## [3,]   -1   -2   -3
```

3 Sudoku (40 Punkte)

Für $n, m \in \mathbb{N}$ bezeichnen wir eine Matrix $S \in \{\texttt{NA}, 1, \ldots, nm\}^{nm \times nm}$ als **Sudoku**. Ist $S \in \{1, 2, \ldots, nm\}^{nm \times nm}$ bezeichnen wir das Sudoku als **ausgefüllt**. Ein Sudoku ist **gültig**, falls in jeder Zeile und in jeder Spalte sowie in jedem Feld jede Zahl in $\{1, \ldots, nm\}$ höchstens einmal vorkommt. Hierbei bezeichnen wir mit **Feld** die nm partitionierenden Teilmatrizen der Größe $n \times m$. Ein ausgefülltes und gültiges Sudoku heißt **gelöst**. Gibt es zu einem Sudoku S ein gelöstes Sudoku S^*, das mit S in allen Einträgen, die nicht `NA` sind, übereinstimmt, so heißt S **lösbar** und S^* **Lösung** von S.

Folgendes Bild stellt ein leeres Sudoku mit $n = 2$ und $m = 3$ dar.



In R repräsentieren wir ein Sudoku mit einer numerischen quadratischen Matrix, die zusätzliche Attribute enthält:

- Das Attribut `class` hat den Wert `"sudoku"`.
- Das Attribut `size` hat den Wert `c(n,m)` mit `n` und `m` wie n, m in der Beschreibung oben.

In den folgenden Aufgaben dürfen keine Schleifen (`for`, `while`, ...) oder Rekursionen benutzt werden. Funktionen der `apply`-Familie dürfen nur in Teilaufgabe 6. benutzt werden. Siehe dazu Crashkurs Abschnitt 16.

1. Schreibe eine Funktion `size(mat)` mit zugehöriger Ersetzungsfunktion, die den Wert des Attributs `size` von `mat` zurückgibt bzw setzt.
2. Schreibe eine Funktion `sudoku(mat, n, m)`, die aus einer quadratischen Matrix `mat` ein Sudoku macht, dh `mat` die entsprechenden Attribute hinzufügt und das entstandene Objekt zurückgibt. Hierbei soll `n` automatisch gesetzt werden, falls nur `m` übergeben wird und anders herum.
3. Schreibe eine Funktion `is_sudoku(s)`, die `TRUE` ausgibt, falls `s` ein Sudoku ist und sonst `FALSE`. Neben der mathematischen Definition vom Anfang müssen dabei auch die Attribute getestet werden.
4. Schreibe eine Funktion `is_sub_valid(x)`, welche `TRUE` ausgibt, falls `x` ein Vektor oder eine Matrix ist (muss nicht getestet werden), die nur Einträge aus `NA`, `1`, `2` ..., `length(x)` hat und Nicht-`NA`-Einträge maximal einmal vorkommen; sonst `FALSE`.
5. Schreibe eine Funktion `partition_index(n, m)`, die eine Matrix $I \in \{1, \ldots, nm\}^{nm \times nm}$ ausgibt, welche in allen Einträgen des i-ten Feldes den Wert i hat, $i = 1, \ldots, nm$. **Hinweis:** Erzeuge zuerst eine Matrix der Form $\tilde{I}$, dann I. In beiden Schritten ist die Funktion `rep()` nützlich.

$$\tilde{I} = \begin{pmatrix} 1 & 4 \\ 1 & 4 \\ 2 & 5 \\ 2 & 5 \\ 3 & 6 \\ 3 & 6 \end{pmatrix}, \qquad I = \begin{pmatrix} 1 & 1 & 1 & 4 & 4 & 4 \\ 1 & 1 & 1 & 4 & 4 & 4 \\ 2 & 2 & 2 & 5 & 5 & 5 \\ 2 & 2 & 2 & 5 & 5 & 5 \\ 3 & 3 & 3 & 6 & 6 & 6 \\ 3 & 3 & 3 & 6 & 6 & 6 \end{pmatrix}$$

6. Schreibe die Funktionen `is_valid(s)`, `is_filled_in(s)`, `is_solved(s)`, die `TRUE` ausgeben, falls das Sudoku `s` gültig, ausgefüllt, bzw gelöst ist; sonst `FALSE`. **Hinweis:** Nutze für `is_valid()` die Funktionen `apply()`, `sapply()`, `partition_index()` und `is_sub_valid()`.
7. Schreibe eine Funktion `is_solution_of(s, s_star)`, die `TRUE` ausgibt, falls `s_star` eine Lösung von `s` ist; sonst `FALSE`.
8. Schreibe eine Funktion `print_non_valid(s, print_missing=TRUE)`, die auf der Konsole die Indizes, Zeilen, Spalten, oder Felder von `s` ausgibt, die `s` ungültig machen, siehe Beispiele unten. Fehlende Einträge werden nur dann ausgegeben, wenn `print_missing` auf `TRUE` (Default) gesetzt ist . Nutze dazu die Funktion `which()` ggf mit dem Argument `arr.ind = TRUE` oder die Funktion `arrayInd()`.

Wenn nicht explizit angegeben, muss nicht auf Korrektheit der Datenstrukturen der Argumente (etwa mit `stopifnot()`) getestet werden (auch wenn dies natürlich sinnvoll wäre...).

Hinweis: potentiell nützliche Funktionen: `rep()`, `sort()`, `all()`, `any()`, `%in%`, `is.na()`, `prod()`, `unique()`, `duplicated()`, `attr()`, `apply()`, `sapply()`, `which()`, `arrayInd()`

```
x <- matrix(
  c(5,2,6,4,3,1,6,1,3,2,5,4,3,4,1,5,2,6,2,6,4,3,1,5,1,3,5,6,4,2,4,5,2,1,6,3),
  ncol=6)
is_sudoku(x)
## [1] FALSE

s <- sudoku(x, 3, 2)
s
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    5    6    3    2    1    4
## [2,]    2    1    4    6    3    5
## [3,]    6    3    1    4    5    2
## [4,]    4    2    5    3    6    1
## [5,]    3    5    2    1    4    6
## [6,]    1    4    6    5    2    3
## attr(,"class")
## [1] "sudoku"
## attr(,"size")
## [1] 3 2
size(s)
## [1] 3 2
is_sudoku(s)
## [1] TRUE
is_filled_in(s)
## [1] TRUE
is_valid(s)
## [1] FALSE
is_solved(s)
## [1] FALSE
print_non_valid(s)
## field 1,1 invalid
## field 2,1 invalid
```

```
## field 3,1 invalid
## field 1,2 invalid
## field 2,2 invalid
## field 3,2 invalid

size(s) <- c(2, 3)
is_sudoku(s)
## [1] TRUE
is_filled_in(s)
## [1] TRUE
is_valid(s)
## [1] TRUE
is_solved(s)
## [1] TRUE

s_na <- s
s_na[sample(36, 18)] <- NA
s_na
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]   NA    6   NA    2    1   NA
## [2,]    2   NA    4    6   NA    5
## [3,]    6   NA   NA   NA   NA    2
## [4,]    4   NA    5   NA    6    1
## [5,]   NA    5    2    1   NA   NA
## [6,]    1   NA    6   NA   NA   NA
## attr(,"class")
## [1] "sudoku"
## attr(,"size")
## [1] 2 3
is_sudoku(s_na)
## [1] TRUE
is_filled_in(s_na)
## [1] FALSE
is_valid(s_na)
## [1] TRUE
is_solved(s_na)
## [1] FALSE
print_non_valid(s_na)
## missing value at 1,1
## missing value at 5,1
## missing value at 2,2
## missing value at 3,2
## missing value at 4,2
## missing value at 6,2
## missing value at 1,3
## missing value at 3,3
## missing value at 3,4
## missing value at 4,4
## missing value at 6,4
## missing value at 2,5
## missing value at 3,5
## missing value at 5,5
## missing value at 6,5
```

```
## missing value at 1,6
## missing value at 5,6
## missing value at 6,6

s_not <- sudoku(sample(1:6, 36, replace=T), 2, 3)
s_not[sample(36, 18)] <- NA
s_not
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]   NA   NA    2    4   NA    4
## [2,]   NA    5    6    4   NA    5
## [3,]    6   NA   NA   NA   NA   NA
## [4,]    5   NA   NA    5    3    2
## [5,]    3    5    1   NA   NA    3
## [6,]   NA   NA    6   NA    4   NA
## attr(,"class")
## [1] "sudoku"
## attr(,"size")
## [1] 2 3
is_sudoku(s_not)
## [1] TRUE
is_filled_in(s_not)
## [1] FALSE
is_valid(s_not)
## [1] FALSE
is_solved(s_not)
## [1] FALSE
print_non_valid(s_not, print_missing=FALSE)
## row 1 invalid
## row 2 invalid
## row 4 invalid
## row 5 invalid
## col 2 invalid
## col 3 invalid
## col 4 invalid
## field 2,2 invalid
```