Winfried Auzinger
Dirk Praetorius

Sommersemester 2016
26. April 2016

Übungsaufgaben zur VU Computermathematik

Serie 7

Exercise 7.1: ***Relations and matrices: some basic operations.***

A relation on a set X is a generalization of a function $f\colon X \to X$. It is a mapping $r\colon X \times X \to \{\texttt{true}, \texttt{false}\}$; x and y are related to each other by r if $r(x,y)$ is true.[7] '$r(x,y)$' simply means that $r(x,y)$ holds.

For instance, $r(x,y) := x < y$ is a relation on $\mathbb{R}$.

Here we consider the case of a finite set $X = \{1, 2, \ldots, n\}$ for fixed $n \in \mathbb{N}$. Then a relation on X can be represented by an $n \times n$ - matrix R with entries `true` or `false` (using 1 and 0 instead may be more convenient). $R[i,j] = \texttt{true}$ mean that $r(i,j)$ holds. For instance, the relation $i = j$ is represented by the identity matrix ($1 = \texttt{true}$ only on the diagonal.).

a) *To build the matrix R for the relation $i < j$ on $X = \{1, 2, \ldots, n\}$ for given $n \in \mathbb{N}$, design an appropriate procedure.*

b) A relation r is called reflexive if $r(i,i)$ for all $i \in X$.

Design a function `isreflexive(R)` *which returns* `true` *if R defines a reflexive relation, and* `false` *otherwise.*

c) A relation r is called symmetric if $r(i,j)$ implies $r(j,i)$.

Design a function `issymmetric(R)` *which returns* `true` *if R defines a symmetric relation, and* `false` *otherwise.*

d) A relation r is called transitive if $r(i,j)$ and $r(j,k)$ imply $r(i,k)$.

Design a function `istransitive(R)` *which returns* `true` *if R defines a transitive relation, and* `false` *otherwise.*

Hint: In R, represent `true` by 1 and `false` by 0 and look at the matrix product R^2. Then it can be shown that R defines a transitive relation iff $R^2[i,j] \neq 0$ implies $R[i,j] \neq 0$.

A reflexive, symmetric and transitive relation is called an equivalence relation.

e) Let X be decomposed into a union of a family of nonempty pairwise disjoint subsets. Then,

$$r(x,y) := x \text{ and } y \text{ are contained in the same subset of this family}$$

is an equivalence relation. Turning this around, each equivalence relation defines such a decomposition of X into subsets – so-called equivalence classes.

Assume that the matrix R represents an equivalence relation r on $X = \{1, \ldots, n\}$. Design a procedure `eqclass(`i,R`)` *which returns the equivalence class*

$$\{j \in X : r(i,j)\} \subseteq X$$

in form of a set. Test an example.

---

[7] The set $\{(x,y) \in X \times X : r(x,y) = \texttt{true}$ is a subset of the Cartesian product $X \times X$. Specifying a subset of $X \times X$ is equivalent to specifying a relation r.

Exercise 7.2: ***Boolean matrix product.***

Let $A = (a_{ij})$ and $B = (b_{ij})$ be square matrices of the same dimension with entries `true` or `false`, as in **7.1**. We call them Boolean matrices. The Boolean matrix product is defined as the matrix $C = (c_{ij})$, with

$$c_{ij} = \texttt{any}(\{(a_{ik} \texttt{ and } b_{kj}),\ k = 1 \ldots n\}),$$

where `any(...)` is `true` if at least one of its arguments is `true` (generalization of `or` for more than two arguments).

a) *Provide an implementation of* `any` *in form of a function or a procedure expecting a Boolean vector (i.e., a vector with entries* `true` *or* `false`*) as its argument.*

b) *Implement Boolean matrix multiplication in form of a procedure, using* `any`*.*

c) *Repeat* **a)** *and* **b)***, but assuming that* `true` *and* `false` *are represented by* 1 *and* 0*, respectively.*

Remark: A Boolean matrix R represents a finite relation. This relation is transitive iff $R^2[i,j] = 1$ implies $R[i,j] = 1$, where R^2 is the Boolean matrix product; see **7.1**.

Exercise 7.3: ***An*** argmin ***implementation.***

- *Design a procedure* `argmin(`A`::{Vector,Matrix})`[8] *which accepts an object A of type* `Vector` *or* `Matrix` *as its argument and returns the position*[9] *of a minimal element in A (minimal in the sense of ordering of $\mathbb{R}$) together with the value of the minimum. For the case of a* `Matrix` *A, the 'position' is the corresponding pair of indices.*

Include a check whether all elements of A have a real numerical value (use `is(...,numeric)`*).*[10] *If one of these tests fails, exit with an* `error` *- message.*

Remark: In Maple, there is `min` but there seems not to exist something like `argmin`.

Hint: Using `type` you can determine the type of an object. In this way you can discern between `Vector` and `Matrix`.

Exercise 7.4: ***Sudoku (i).***

We represent a classical 9×9 Sudoku by a `Matrix`[11] S and begin to play. To find a single correct entry in a given incomplete S (example):

$S =$

			2			3	9	7
8			3		9		6	
	3		1			4		
3	1	5						4
	8		6	3	5		1	
2						5	3	8
		9			6			
	4		7		3			6
5	6	3			8			

realize the following operations:

[8] This syntax means that arguments of the type `Vector` or `Matrix` are accepted; otherwise the procedure will automatically exit with an error message (try). For accepting a single type only, e.g., `Vector`, one would use the syntax A`::Vector`.

[9] The minimal value may be attained several times, but returning only one of them is required here.

[10] Data types are organized in a hierarchic way. E.g., the types `integer`, `rational`, `float` are sub-types of the type `numeric` representing any numerical real value.

[11] Empty positions are represented by 0.

a) *Design a procedure* `iscorrect(`*i,j,n,S*`)` *which checks whether inserting* n $\in \{1,2,3,4,5,6,7,8,9\}$ *at position* (i,j) *is correct, i.e., it observes the rules (no double entry in line, column, or surrounding* 3×3 *box). Your procedure returns* `true` *or* `false`*, respectively.*

This is of course not enough; for instance, in the above example, inserting 2, 5, 7 or 9 instead of 3 at position (3,2) would be also correct.

b) *Modify your procedure from* **a)***, checking whether the entry* n *must necessarily be inserted at position (i,j). This means that*

- *either in line i,*
- *or in column j,*
- *or in the surrounding* 3×3 *box,*

there is no other location where n *can be inserted. In this case, also return a copy of S with the entry correctly inserted.* [12]

In the example, 3 is necessarily to be inserted at position (3,2) because all other positions in the upper left 3×3 box are blocked.

Exercise 7.5: (∗) ***Sudoku (ii).***

• *Design a procedure for solving a Sudoku puzzle, using* **7.4 b)***. Use a brute-force strategy scanning all empty fields and trying all possibilities.*

Now, two cases can occur::

- In very simple cases, you always find an entry for which there is no alternative according to **7.4 b)**, and the process successfully runs to completion.
- Usually, at some point you do not find such an entry, and your process stops at this point.

  In such a case, some more refined look-ahead strategy is required, but such a more complicated algorithm is not the topic of this exercise.

Test what happens for the above example – it has low degree of difficulty. For more difficult examples, the behavior will be different.

Exercise 7.6: ***Nothing special; just to train recursion.***

a) Devils's staircase.

Consider the sequence of continuous functions [13] $f_n\colon\, [0,1] \to [0,1]$, recursively defined by $f_0(x) := x$ and

$$f_n(x) := \begin{cases} \frac{1}{2}\, f_{n-1}(3\,x), & 0 \le x < \frac{1}{3}, \\ \frac{1}{2}, & \frac{1}{3} \le x \le \frac{2}{3}, \\ \frac{1}{2}\left(1 + f_{n-1}(3\,x - 2)\right), & \frac{2}{3} < x \le 1 \end{cases}$$

for $n \ge 1$.

Implement these functions in form of a recursive function or procedure, and produce plots for several values of n.

b) Let A and B be $N \times N$ matrices, N even. By partitioning A and B into four blocks of dimension $\frac{N}{2} \times \frac{N}{2}$, the matrix product $A \cdot B$ can be realized using 8 'smaller' matrix multiplications and 4 smaller matrix additions. [14]

Assume that $N = 2^n$*, and use partitioning in a recursive way to compute the matrix product* $A \cdot B$*. Realize this in form of a recursive procedure* `rmp(`*A,B*`)`*.*

[12] The matrix S could also be a global variable.

[13] Reemark: The limiting function $\lim_{n\to\infty} f_n(x)$ exists, it is continuous and nowhere differentiable.

[14] Actually, it can be shown that one of these 8 multiplications can be replaced by a few additions. This observation is the basis for more efficient recursive algorithms ('Strassen algorithm').

Exercise 7.7: ***Formatted output.***

a) *Design a procedure* `print_sudoku(`S`)`, *which writes a formatted output of a Sudoku (see* **7.4**)*) to the screen:*

```
+++++++++++++++++++++++++
+   8   + 4 2 3 +     3 +
+ 8 2 3 +   2 7 + 1   5 +
+ 9   2 + 1 5 3 +       +
+++++++++++++++++++++++++
+ 1 2   +   2 7 + 5   3 +
+     3 + 7   3 + 1 2   +
+ 1   3 + 5 2 8 + 1   2 +
+++++++++++++++++++++++++
+ 1 2 8 + 4 2 3 + 7   9 +
+ 7   3 + 4   3 +       +
+   2 5 +   2 9 + 1   3 +
+++++++++++++++++++++++++
```

Use an auxiliary function which converts 0 to the string `" "` and integers $n > 0$ to the string `"`n`"`.

Hint: Use `sprintf` and `printf`.

b) *Design a procedure* `print_sudoku(`*S,filename*`)` *which prints a Sudoku to a textfile (the filename is specified as a string).*

Hint: Use `fprintf`.

Exercise 7.8: ***Your favorite package?***

Look at the help page `? index`, and select `packages.` Here you see a complete list of available packages.

- *Choose one of them, have a closer look, and prepare a small demo of its basic features.*

There are many different packages. If you have no other special preference, you may take a closer look at the `plots` and `plottools` packages. The package `geometry` is also very nice. *Aficionados* of combinatorics may look at `combinat` (also `combstruct`). ... ...