*Institut für Theoretische Informatik*
*Dr. B. Gärtner, Prof. Dr. J. Hromkovič*

*15. November 2011*

# Informatik für Mathematiker und Physiker Serie 8 HS 11

URL: http://www.ti.inf.ethz.ch/ew/courses/Info1_11/

## Aufgabe 1 (4 Punkte)

Consider the problem of finding a shortest robot path from the lecture. In reality, the robot has to turn when the path makes a turn (by 90 degrees clockwise or counterclockwise), and this takes time. Suppose therefore that each turn of the robot requires one extra step. In this setting, the path on page 157 of the lecture notes becomes longer by 7 (the number of turns). In particular, the path is not a shortest one anymore in this new model (can you find a shorter one)? How would you extend the method described in the lecture to compute shortest robot paths in the new model where each turn requires one extra step? This is a theory exercise.

## Skript-Aufgabe 81 (4 Punkte)

Write a program `inverse_matrix.cpp` that inverts a $3 \times 3$ matrix $A$ with real entries. The program should read the nine matrix entries from the input, and then output the inverse matrix $A^{-1}$ (or the information that the matrix $A$ is not invertible). In addition, the program should output the matrix $AA^{-1}$ in order to let the user check whether the computation of the inverse was accurate (in the fully accurate case, the latter product is the identity matrix).

**Hint**: For the computation of the inverse, you can employ *Cramer's rule*. Applied to the computation of the inverse, it yields that $A^{-1}_{ij}$ (the entry of $A^{-1}$ in row $i$ and column $j$) is given by

$$A^{-1}_{ij} = \frac{(-1)^{i+j}\det(A^{ji})}{\det(A)},$$

where $\det(M)$ is the determinant of a square matrix $M$, and $A^{ji}$ is the $2 \times 2$ matrix obtained from $A$ by deleting row $j$ and column $i$.

To compute the determinant of a $3 \times 3$ matrix, you might want to use the well-known *Sarrus' rule*.

## Skript-Aufgabe 85 (4 Punkte)

Consider the string matching algorithm of `string_matching.cpp`. Prove that for all $m > 1, n \geq m$, there exists a search string $s$ of length $m$ and a text $t$ of length $n$ on which the algorithm in `string_matching.cpp` performs $m(n-m+1)$ comparisons between single characters.

## Skript-Aufgabe 87 (4 Punkte)

Write a program `frequencies.cpp` that reads a text from standard input (like in `string_matching.cpp`) and outputs the frequencies of the letters in the text, where we do not distinguish between lower and upper case letters. For this exercise, you may assume that the type `char` implements ASCII encoding. This means that all characters have integer values in $\{0, 1, \ldots, 127\}$. Moreover, in ASCII, the values of the $26$ upper case literals `'A'` up to `'Z'` are consecutive numbers in $\{65, \ldots, 90\}$; for the lower case literals `'a'` up to `'z'`, the value range is $\{97, \ldots, 122\}$.

Running this on the lyrics of *Yesterday* (The Beatles) for example should yield the following output.

```
Frequencies:
a:      45 of 520
b:      5 of 520
c:      5 of 520
d:      28 of 520
e:      65 of 520
f:      4 of 520
g:      13 of 520
h:      27 of 520
i:      27 of 520
j:      0 of 520
k:      3 of 520
l:      20 of 520
m:      10 of 520
n:      30 of 520
o:      43 of 520
p:      4 of 520
q:      0 of 520
r:      19 of 520
s:      36 of 520
t:      31 of 520
u:      9 of 520
v:      6 of 520
w:      19 of 520
x:      0 of 520
y:      34 of 520
z:      0 of 520
Other: 37 of 520
```

Die **Aufgaben 89** und **90** aus den Vorlesungsunterlagen sind die **Challenge Aufgaben** und geben jeweils 8 Punkte, wenn sie vollständig gelöst werden.

**Abgabe**: Bis 22. November 2011, 15.15 Uhr.