Name: Matriculation Number:

Final Exam
General CS II (320102)

May 25., 2014

You have two hours(sharp) for the test;
Write the solutions to the sheet.

The estimated time for solving this exam is 0 minutes, leaving you 120 minutes for revising your exam.

You can reach 0 points if you solve all problems. You will only need 100 points for a perfect score, i.e. -100 points are bonus points.

Different problems test different skills and knowledge, so do not get stuck on one problem.

	To be used for grading, do not write here	
prob.	Sum	grade
total	0	
reached		

Please consider the following rules; otherwise you may lose points:

- "Prove or refute" means: If you think that the statement is correct, give a formal proof. If not, give a counter-example that makes it fail.
- Always justify your statements. Unless you are explicitly allowed to, do not just answer "yes" or "no", but instead prove your statement or refer to an appropriate definition or theorem from the lecture.
- If you write program code, give comments!

1 Graphs & Trees

Conjecture 1.1 *Given a tree $G = \langle V, E\rangle$ with root node v_r. For any node $v \in V$ except the root v_r, there is exactly one path $p \in \Pi(G)$ with start$(p) = v_r$ and end$(p) = v$.*

Problem 1.1 (Unique Paths in Trees)
Prove the conjecture above by induction on the number of nodes or refute it by a counterexample. 10pt 10min

Solution:
Proof: by induction over $n = \#(V)$

P.1.1 $n = 1$ **(base case)**: $V\backslash\{v_r\} = \emptyset$, i.e. there is no node except the root node, and thus we have nothing to prove. □

P.1.2 $n \to n + 1$ **(induction step)**: Let $v \in V$ be a non-root node. From the definition of a tree, it follows that indeg$(v) = 1$, and thus there is exactly one $u \in V$ with $\langle u, v\rangle \in E$. As $\#(V\backslash\{v\}) = n$ we can infer from the inductive hypothesis that there is exactly one path $p \in \Pi(G)$ with start$(p) = v_r$ and end$(p) = u$. If we append the edge $\langle u, v\rangle$ to that path, we obtain a unique path $q = \langle v_r, \ldots, u, v\rangle$ with start$(q) = v_r$ and end$(q) = v$ □

□

---

2 Circuits and Positional Number Systems

Problem 2.1 (Gray Code)
A Gray code is a number encoding in which every two consecutive numbers differ by a single bit only. Below there is an example of a 2-bit Gray code: 15pt 15min

Gray encoding	binary equivalent	decimal equivalent
00	00	0
01	01	1
11	10	2
10	11	3

- Design a 4-bit Gray code. You can provide your answer in a form similar to the table above. Keep in mind that every two consecutive numbers must differ by the value of only one of the four bits.
- Create a combinatorial circuit that takes a 4-bit number in Gray code as input, and outputs its 4-bit binary equivalent.

Solution:

- There are various solutions for a 4-bit Gray code. One of them is the following:

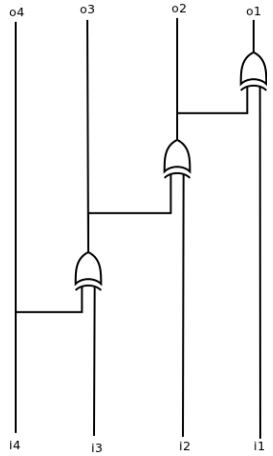
Gray encoding	binary equivalent	decimal equivalent
0000	0000	0
0001	0001	1
0011	0010	2
0010	0011	3
0110	0100	4
0111	0101	5
0101	0110	6
0010	0111	7
1100	1000	8
1101	1001	9
1111	1010	10
1110	1011	11
1010	1100	12
1011	1101	13
1001	1110	14
1000	1111	15

- Let us name the input bits i_1, i_2, i_3 and i_4, with i_1 being the least significant bit. Similarly, the output bits would be o_1, o_2, o_3 and o_4.

  By comparing the first two columns of the table above, we note the following:

  - $o_4 = i_4$
  - $o_3 = i_3$ if $i_4 = 0$ and $o_3 = \overline{i_3}$ if $i_4 = 1$. Thus $o_3 = XOR(i_3, i_4)$
  - $o_2 = i_2$ if $i_3 = i_4$, i.e. if $o_3 = 0$, and $o_2 = \overline{i_2}$ if $i_3 = \overline{i_4}$, i.e. if $o_3 = 1$. Thus $o_2 = XOR(i_2, o_3)$
  - $o_1 = i_1$ if there is an odd number of 0's among i_2, i_3 and i_4, and $o_1 = \overline{i_1}$ otherwise. Thus $o_1 = XOR(o_2, i_1)$

  So the corresponding circuit is:



Problem 2.2 (Number System Conversion)

Convert the following base-3 numbers into octal and decimal numbers: 5pt

5min

1. 11
2. 121220
3. 1010

Then convert the following binary numbers to base-5:

1. 10101
2. 111

3 Machine Languages

Problem 3.1 (Count the Contestants) 12pt 12min

You have just entered a new contest and got a decent score. However, you are afraid that you might not qualify to the next stage, so you quickly write an `ASM` program to help you figure how many contestants will advance.

You are given all the N scores, in a decreasing order and the cutoff score k. All participants with a score lower than the cutoff will not advance. You may assume that all scores are positive.

You are given k in $D(1)$ and $N > 3$ values representing the scores of the participants in $D(2), \ldots, D(N+1)$ (N is not given, but you may rest assured that $N > k$). You must output the requested value in $D(0)$.

Example: $N = 8$, $k = 5$ and the scores $(10, 9, 8, 7, 7, 7, 5, 5)$. Your program should stop with $D(0) = 6$.

Hint: You are allowed to overwrite any position in memory, if needed!

Note: Write down your idea first! It will be more valuable than the code itself.

Solution: The following C snippet might clear up the solution:

```
int nr = 0; /* the number of contestants that advance, in D(0) */
int dk = D[k+1]; /* store this value in D(1) since we do not really
                    need k */
while ( D[nr+2] >= dk && D[nr+2] > 0 ) /* we iterate using nr */
      nr ++;
return nr;
```

Since there is no logical AND operation in ASM, we will use nested if constructs (plus comparisons with 0), such as:

```
if ( D[nr+2] - dk >= 0 )
  if ( D[nr+2] > 0 )
    /* ... */
```

The `ASM` code follows:

```
LOAD 1
MOVE ACC IN2
LOADI 0
STORE 0
LOAD 0
MOVE ACC IN1
LOADIN 2 1
STORE 1
LOADIN 1 2
SUB 1
JUMP(<) 7
LOADIN 1 2
JUMP(<=) 5
LOAD 0
ADDI 1
STORE 0
JUMP −12
STOP 0
```

Problem 3.2 (Perfect Squares in $\mathcal{L}(\texttt{VMP})$) 15pt

Write an $\mathcal{L}(\texttt{VMP})$ procedure that, given a number n as the only parameter, returns 1 if n is a perfect square and 0 otherwise. 15min

(Recall: an integer n is a perfect square if there exists another integer x such that $n = x \cdot x$.)

Additionally, please provide the same procedures in μML as well.

Hint: You may want to define additional procedures.

Solution:

```
proc 2 32,                              // check(a, b) for checking (a == b * b) → call by 'call 0
arg 1, arg 2, arg 2, add, leq, cjp 15,  // if(2b > a) jump to RETURN0
arg 2, arg 2, mul, arg 1, sub, cjp 15,  // if(b * b == a) jump to RETURN1

con 1, arg 2, add, arg 1, call 0, return,  // otherwise we go on incrementing b
                                           // we call check(a, b + 1)
con 0, return,                          // RETURN0 → a is not a perfect square
con 1, return,                          // RETURN1 → a is a perfect square

proc 1 33,                              // isSquare(a) → to be called by call 'call 32'
arg 1, cjp 25,                          // if(a == 0) jumps to RETURN_1;
con 1, arg 1, sub, cjp 18,              // if(a == 1) jumps to RETURN_1;
arg 1, con 0, leq, cjp 8,               // if(0 > a) jumps to RETURN_0;
con 2, arg 1, call 0                    // here a ≥ 2; calling check(a, 2)

con 0, return,                          // RETURN_0 → a is not a perfect square
con 1, return,                          // RETURN_1 → a is a perfect square


con 9, call 32, halt                    // calling it on 9
```

4 Turing Machines

Problem 4.1: Design a Turing machine that checks whether a number written in base 4 is a power of 2 or not. You can assume that the input will be valid and in between # signs. Your Turing machine needs to halt in the state "yes" if the input is a power of 2 and in the state "no" if not. There is no need to define the states. Also, you can overwrite the input if you find it necessary. 15pt 15min

Hint: Think of other bases for numbers, transforming the input may help.

Note: If you have a number of states that are similar in execution you are allowed to omit them from the action table under the condition that you provide a sufficient explanation for how they work.

5 Internet/WWW/XML

Problem 5.1 (Internet Alphabet Soup)

Name/expand and explain in one sentence the significance of following concepts: 5pt 5min

1. URL/ URI/URN
2. WWW
3. HTTP
4. HTML
5. CSS

Solution:

1. URL/ URI/URN: Uniform Resource Locator/Identifier/Name. A URI identifies a data resource on the Internet, a URL also locates it, and a URN is a URI that is not a URL.
2. WWW: World Wide Web. The WWW is the hypertext/multimedia part of the Internet.
3. HTTP: Hypertext Transfer Protocol. The HTTP protocol specifies the way computers exchange
4. HTML: Hypertext Markup Language. HTML is the markup format for web pages on the WWW.
5. CSS: Cascading Style Sheets. CSS is the language for marking up the appearance of web pages that have been marked up functionally.

Problem 5.2 (Packets and Protocols)

When you click on a link in your web browser, packets of data are transmitted to the web-server. The server handles those data packets accordingly and sends back another stream of data packets that are in turn received by your browser and interpreted accordingly. Explain this process in detail, highlighting the role of each layer in the protocol stack plays in data transmission. 8pt 8min

Solution: The packet goes down from the application layer on A via the transport- and internet layers to the link layer. Then on each routing hop it goes up to the Internet layer, where

– based on the destination IP address – it is decided where to route it. At the destination B it goes up from the link layer through internet and transport layers to the application layer. The graph can be seen in the course notes.

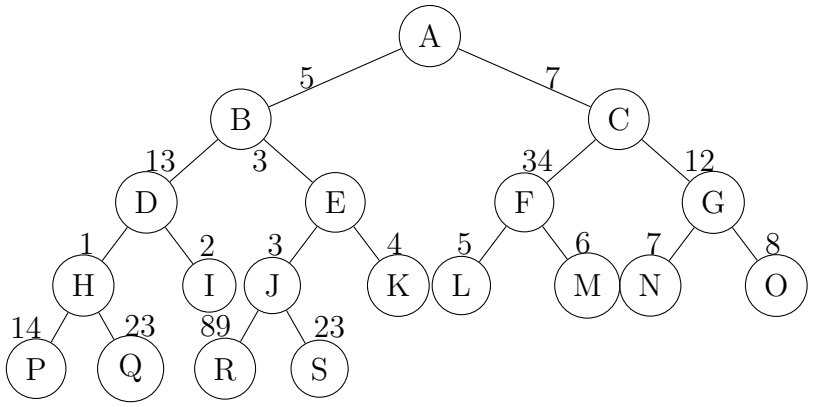
6 Problem Solving and Search

Problem 6.1 (Greedy Search vs. A* Search)

10pt
10min

1. Compare and contrast Greedy Search and A* search (i.e. highlight the differences and the common concepts). What is the role of the heuristic in these two strategies?
2. Apply Greedy and A* on the tree below, where the edge labels are step costs. The heuristic is given in the table on the left. The result should be a list of nodes in the order traversed.

A	40	N	125
B	35	O	126
C	34	P	25
D	20	Q	0
E	30	R	122
F	34	S	55
G	115	T	19
H	14	U	51
I	25	V	78
J	30	W	67
K	70	X	56
L	33	Y	34
M	32	Z	12



Solution: Greedy: ACFLMBDHQ Astar: ACBDHQ

7 Programming in Prolog

Problem 7.1 (Sortin `ProLog` Lists)

12pt
12min

Write a `ProLog` program that sorts a list of integers; i.e. a predicate `sort(L1,L2)` that evaluates to true iff the list `L2` is the sorted permutation of `L1`. For instance, the query

```
?- sort([5,2,6,1],X)
```

should succeed with the answer substitution `X=[1,2,5,6]`.

Hint: Insertion sort is the easiest to implement. First write a function that inserts an element in an already sorted list and then write the actual sort function. You might need another helper function for the sort.