Midterm Grand Tutorial
General CS II (320201)

Spring 2013

1 Graphs and Trees

0pt

Problem 1.1 (Graph theory)

1. Magic carpets

   In Never-Never-Land there is only one means of transportation: magic carpet. Twenty-one carpet lines serve the capital. A single line flies to Farville, while every other city is served by exactly 20 carpet lines.
   Prove or refute the possibility of traveling by magic carpet from the capital to Farville (possibly by transferring from one carpet line to another).

2. Bridges of Konigsburg

   The city of Konigsburg lies on both banks of a river and on two islands in the river. There are seven bridges connecting various parts of the city: two connecting each bank to one of the islands, one bridge between the islands, and one bridge connecting each bank to the other island.
   Prove or refute whether one can walk around Konigsburg, crossing each bridge exactly once.

3. Connectivity

   Prove that a graph with n vertices, each of which has degree at least $\frac{n-1}{2}$ is connected.

2 Combinatorial Circuits

0pt

Problem 2.1 (Reversible Counter)
Build a circuit which simulates a 2-bit reversible binary counter. Your circuit will have 2 inputs and 2 outputs. Here are the input details:

- If the 1st input is 1, your counter will behave like a normal counter, outputting 00, 01, 10, 11 and looping back. However, if the input is 0, your counter will count backwards, i.e. 11, 10, 01, 00 and so on. Note that this value can change while testing your reversible binary counter.

- The 2nd input will be a standard clock signal of a constant frequency. This will be used to set the frequency of your counter.

The output will be the standard 2 bits which represent the current value of the counter. You may use any logical setup which was taught in the slides, as long as you explain its input and output.

Example: If the 1st input is 1 for the first two runs of the clock, and 0 after wards, your output should be 01, 10, 01, 00, 11, 10, etc.

Hint: Use adders and subtracters to increment or decrement the counter.

3 Machine Programming

0pt

Problem 3.1 (Assembler product)

You are given four strictly positive numbers a, b, c and d, which represent the fractions $\frac{a}{b}$ and $\frac{c}{d}$. The four numbers can be found in memory in cells $P(0..3)$. You are required to output in $P(4)$ and $P(5)$ two numbers e and f, such that $\frac{a}{b} + \frac{c}{d} = \frac{e}{f}$. You are not required to simplify the fraction.

Example: $P(0) = 1$, $P(1) = 2$, $P(2) = 1$, $P(3) = 3$; your program should halt with $P(4) = 5$, $P(5) = 6$.

Hint:

$$\frac{a}{b} + \frac{c}{d} = \frac{a \cdot d + b \cdot c}{b \cdot d}$$

0pt

Problem 3.2 (Bit-wise xor in $\mathcal{L}(\texttt{VM})$)

Given two natural numbers x and y, write an $\mathcal{L}(\texttt{VM})$ program which computes the bit-wise exclusive or (xor) function on the numbers' binary* representations (B(x) and B(y)) and converts the result back into a natural number.

Note that we are dealing with natural numbers and we don't care about negative values. If you are in doubt about the encoding please refer to slide 241 on page 148 of the lecture notes, entitled "Arithmetic Circuits for Binary Numbers".

Since you are given x and y, assume your program can begin with "con x con y". Though this is not a valid $\mathcal{L}(\texttt{VM})$ set of instructions, we assume the user replaces the sequences "x" and "y" by the decimal representations of the values of x and y respectively, before running the program through the virtual machine.

Assuming the stack was initially empty (before pushing the values of x and y), make sure that after the execution the stack only contains the result. This is a healthy precondition and we want you to learn good practice. Though it is mandatory, try not to stress this too much, and focus on the algorithm itself. You can deal with cleaning up the stack once you get to the result.

Note: General requirement when writing $\mathcal{L}(\texttt{VM})$ programs:

You may use tags in your $\mathcal{L}(\texttt{VM})$ program to save some (counting) time. The proper way to do use tags is by respecting the following rules:

1. a tag is a string of lower-case letters of the English alphabet enclosed by the symbol "$\langle$" to the left and "$\rangle$" to the right.

2. place the tag before the instruction one would want to jump to when calling "jp" or "cjp". This is called a tag declaration and can only occur once for each tag. Any other occurrence of the tag has to be preceded by either of the instructions "jp" or "cjp" as described in the next point

3. place the tag after "jp" and "cjp" and omit writing the relative jump distance. This is called a tag reference.

4. if (for any reason) you wish to use both tags and relative jumping, take into account that tag declarations are not instructions and should not be counted, whereas tag references (which essentially replace numbers) should be counted.

For reference, you are given here the set of $\mathcal{L}(\texttt{VM})$ instructions. Note that some instructions require an integer number to follow them, which is also counted as instructions: con, add, sub, mul, leq, jp, cjp, halt, peek, poke

Write your program clearly. Should you wish, you may write additional code in a higher level language (HLL), but make sure that the layout is clean. Though you will not receive any points for that, it **might** help the grader understand your solution better and give you the points you deserve. Please note that correct a or partially correct solution in a HLL does not bring you points. Any argument stating that your intention was good (provable by the HLL code), but you failed the implementation (verifiable by the $\mathcal{L}(\texttt{VM})$ program) is invalid. Only the correctness of the $\mathcal{L}(\texttt{VM})$ program will be assessed.

0pt

Problem 3.3 (Perfect numbers and $\mathcal{L}(\texttt{VMP})$)

Write a $\mathcal{L}(\texttt{VMP})$ static procedure that takes 1 number and returns 1 if it is a perfect number or 0 otherwise. A perfect number is a positive integer that is equal to the sum of its proper positive divisors. For example, 6 is a perfect number because it is divisible by 1, 2 and 3 and $1 + 2 + 3 = 6$.

0pt

Problem 3.4 (Conversion with TMs)

Design a TM which converts a binary number into hexadecimal. Use the alphabet {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}.