# Midterm Grand Tutorial
# General CS II (320102)

Spring 2015

## 1 Graphs and Trees

**Problem 1.1 (Planar Graphs)**

A graph $G$ is called planar if $G$ can be drawn in the plane in such a manner that edges do not cross elsewhere than vertices. The geometric realization of a planar graph gives rise to regions in the plane called faces; if $G$ is a finite planar graph, there will be one unbounded (i.e. infinite) face, and all other faces (if there are any) will be bounded. Given a planar realization of the graph $G$, let $v = \#(V)$, $e = \#(E)$, and let $f$ be the number of faces (including the unbounded face) of $G$'s realization. 15pt 15min

Prove or refute the Euler formula, i.e. that $v - e + f = 2$, must hold for a connected planar graph.

---

**Solution:**

**Proof**: Proof by induction on the number of faces

**P.1** If $G$ has only one face, it is acyclic and connected, so it is a tree and $e = v - 1$. Thus $v - e + f = 2$.

**P.2** Otherwise, choose an edge $e$ connecting two different faces of $G$, and remove it; $e$ can then only appear once in the boundary of each face, so the graph remains connected – any path involving $e$ can be replaced by a path around the other side of one of the two faces. This removal decreases both the number of faces and edges by one, and the result then holds by induction. □

---

## 2 Combinatorial Circuits

**Problem 2.1 (Carry Chain and Conditional Sum Adder)**

Determine depth and cost of a 4-bit Carry Chain Adder, a 4-bit Conditional Sum Adder, and a combination of both where two 2-bit Carry Chain Adders are connected with by one Mux. 0pt 15min

Which adder has lowest cost and depth respectively?

---

**Hint:** You don't need to draw the adders. On the other hand don't supply just the result numbers of your cost calculations, since the revisor can differentiate between fundamental and oversights only if he or she can reconstruct the calculation to some extent. Without any comments the wrong result leads to zero points though just a minor mistake was the actual reason.

---

## 3 Machine Programming

**Problem 3.1 (Paving the Kingdom)**

The King of Far Far Away decided that time has come to pave the streets of his kingdom and he decided to assign the planning phase of the project to you, his Virtual Prince Charming! 0pt

You hastily grab a map of the kingdom, given as $N$ integer values representing the heights of the land. You know that the king will be pleased if all the negative heights will be paved, and that the positive heights are not to be touched.

You have to implement an ASM program that, given $N$ in $D(1)$ and the heights of the map in $D(2 \ldots N+1)$, will output the requested value in $D(0)$.

**Hint:** Try to forget about Prince Charming and the Kingdom of Far Far Away when you solve the problem!

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

**Solution:** Translated to plain English: you are given a vector of $N$ integers, count how many are negative. Code follows:

```
LOADI 0
STORE 0
LOAD 1
ADDI 1
MOVE ACC IN1
MOVE IN1 ACC
SUBI 2
JUMP(<=) 10
LOADIN 1 0
JUMP(>) 4
LOAD 0
ADDI 1
STORE 0
MOVE IN1 ACC
SUBI 1
MOVE ACC IN1
JUMP −11
STOP 0
```

### Problem 3.2 (Missing Number)

12pt

Given that a natural number $n > 1$ is stored in S(0) and that on top of it there are $n-1$ **unique** numbers from the set $\mathcal{S} := \{1, 2, ..., n\}$ stored in S(1), S(2), ..., S($n$-1) (i.e. there are no repetitions) find out which number from $\mathcal{S}$ was initially missing on the stack and store it in S(0). 12min

**Solution: Idea:** We would like to compute the sum of the numbers on the stack and then subtract this number from the formula we all know for: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **Two minor details on the way:**

We need to first make ourselves a slot in the memory (like S(1)) where to store the index counter while computing the sum. This involves emptying S(1) after adding it to S(2), such that we don't corrupt the information.

We will also come up with a way to compute $\frac{n(n+1)}{2}$ halfway there and finally store the result in S(0).

Sounds like a good plan! Let's see how it works out:

```
con 0,                          // S(n) = 0; justification: if n was 2, then S(1) would be the only entry
peek 1, peek 2, add, poke 2,    // S(2) += S(1)
con 2, peek 0, sub, poke 1,     // S(1) = n - 2; we need (n-2) iteration
                                // to sum up the (n-1) numbers we've got

peek 1, cjp 12,                 // LOOP_START; jumps when 0 to LOOP_END
add,                            //
con 1, peek 1, sub, poke 1,     // S(1)--; decrementing counter
jp -12,                         // jumping back to LOOP_START

peek 2, add,                    // LOOP_END; the sum is in S(2); we meanwhile duplicate it
peek 0, con 1, peek 0           // we duplicated S(2) and
add, mul, sub,                  // subtracted it from n(n+1)

poke 1,                         // we go one cell lower; we have the doubled result in S(1)
                                // now really handy: n = S(0) is the biggest result we could have

peek 0, peek 0, add             // LOOP_2_START; we push 2*S(0)
peek 1, sub, cjp 11,            // when 0 we got a 'hit' → jumps to STOP
con 1, peek 0, sub, poke 0      // n--;
jp -17,                         // jumping back to LOOP_2_START;

cjp 2,                          // STOP; performing optional clean-up
                                // we pop() S(1) using some useless cjp statement
halt                            // we're done
```

**Problem 3.3 (Least Common Multiple in $\mathcal{L}(\texttt{VMP})$)**

10pt

10min

Write some $\mathcal{L}(\texttt{VMP})$ procedure(s) which can compute the least common multiple of two positive natural numbers and call it to find out the least common multiple of 4 and 6.

**Hint:** The LCM of two numbers $a$ and $b$ satisfies the following identity:
$LCM(a,b) * GCD(a,b) = a * b$

**Solution:** We will use the formula $lcm(a,b) = \dfrac{a \cdot b}{gcd(a,b)}$ and the euclidian algorithm for computing $gcd(a,b)$ using repeating subtractions:

```
proc 2 44,                                  // procedure for gcd(a, b) → to be called with 'call 0'
arg 1, cjp 33                               // jumps to RETURN2
arg 2, cjp 32                               // jumps to RETURN1
arg 2, arg 1, leq, cjp 12                   // jumps to CASE_1_BIGGER
arg 1, arg 2, sub, arg 1, call 0, return    // CASE_2_BIGGER; we return gcd(a, b − a)
arg 2, arg 2, arg 1, sub, call 0, return    // CASE_1_BIGGER; we return gcd(a − b, b)
arg 2, return                               // RETURN2; returns b (when a == 0)
arg 1, return                               // RETURN1; returns a (when b == 0)

proc 2 26                                   // procedure for a DIV b → to be called with 'call 44'
arg 1, arg 2, leq, cjp 5                    // jumps to CASE_GO
con 0, return                               // CASE_RETURN; we return 0 (since a ¡ b)
con 1, arg 2, arg 2, arg 1, sub,            // CASE_GO; we return 1 + ((a − b) DIV b)
call 44, add, return                        //

proc 2 34                                   // procedure for lcm(a, b) → to be called with 'call 70'
arg 1, con 1, leq, cjp 23                   // jumps to ERROR when a < 1
arg 2, con 1, leq, cjp 16                   // jumps to ERROR when b < 1
arg 2, arg 1, call 0,                       // calling for gcd(a, b)
arg 2, arg 1, mul, call 44, return          // RETURN; we return (a · b) DIV gcd(a, b)
con -1, return,                             // ERROR: we're only supposed to deal with positive numbers

con6, con 4, call 70, halt                  // calling the lcm on 4 and 6 procedure
```

**Problem 3.4 (Integer Division in SW)**

Write a SW program using Concrete Syntax, respective Abstract Syntax, which takes two decimal numbers $a$ and $b$ and computes the value of $ab$ (where represents the decimal division). For example, for $a = 512$ and $b = 5$, the output should be 102. 10pt 10min

**Solution:** Concrete syntax:

```
var a:=512; var b:=5; var div:=0;
while ( a>=b ) do
    a := a−b;
    div := div+1;
end;
return div;
```

Abstract syntax:

```
([("a", Con 512), ("b", Con 5), ("div", Con 0)],
While( Leq(Var "b", Var "a"),
    Seq [
        Assign(Var "a", Sub(Var "a", Var "b")),
        Assign(Var "div", Add(Var "div", Con 1))
    ]
),
Var "div")
```

**Problem 3.5** Write a $\mu$ML program that, given a natural number $n$, computes the $n^{th}$ Fibonacci number. The Fibonacci numbers are recursively defined as: $f(0) = 0, f(1) = 1, f(n) = f(n-1) + f(n-2)$. 10pt 10min

To receive full points, the procedure has to run in linear time, i.e. for any input $n$, at most $cn$ operations are executed where $c$ is some constant. Otherwise, a solution that correctly computes the Fibonacci numbers will receive 70 of the points for this problem.

Assume that $n = 5$ in your program.

**Hint:** You should write a helper procedure.
A simple solution can be written which for input $n$ does exactly $n + 1$ procedure calls.

# 4 Turing Machines

**Problem 4.1** 12pt 10min

Given a tape filled with ones and zeroes find search the tape for the occurrence of the binary representation of the number 45.
If the number 45 is found the turing machine deletes all input and prints a 1.
If the number 45 is not found the turing machine deletes all input and prints a 0. Define a transition table.

**Solution:**

| q | r | w | s | n |
|---|---|---|---|---|
| $q_0$ | 0 | 0 | $q_0$ | $\rightarrow$ |
| $q_0$ | 1 | 1 | $q_1$ | $\rightarrow$ |
| $q_0$ | # | # | $q_{del0}$ | $\leftarrow$ |
| $q_1$ | 0 | 0 | $q_{10}$ | $\rightarrow$ |
| $q_1$ | 1 | 1 | $q_1$ | $\rightarrow$ |
| $q_1$ | # | # | $q_{del0}$ | $\leftarrow$ |
| $q_{10}$ | 0 | 0 | $q_0$ | $\rightarrow$ |
| $q_{10}$ | 1 | 1 | $q_{101}$ | $\rightarrow$ |
| $q_{10}$ | # | # | $q_{del0}$ | $\leftarrow$ |
| $q_{101}$ | 0 | 0 | $q_{10}$ | $\rightarrow$ |
| $q_{101}$ | 1 | 1 | $q_{1011}$ | $\rightarrow$ |
| $q_{101}$ | # | # | $q_{del0}$ | $\leftarrow$ |
| $q_{1011}$ | 0 | 0 | $q_{10110}$ | $\rightarrow$ |
| $q_{1011}$ | 1 | 1 | $q_1$ | $\rightarrow$ |
| $q_{1011}$ | # | # | $q_{del0}$ | $\leftarrow$ |
| $q_{10110}$ | 0 | 0 | $q_0$ | $\rightarrow$ |
| $q_{10110}$ | 1 | 1 | $q_{101101}$ | $\rightarrow$ |
| $q_{10110}$ | # | # | $q_{del0}$ | $\leftarrow$ |
| $q_{101101}$ | 0 | 0 | $q_{101101}$ | $\rightarrow$ |
| $q_{101101}$ | 1 | 1 | $q_{101101}$ | $\rightarrow$ |
| $q_{101101}$ | # | # | $q_{del1}$ | $\leftarrow$ |
| $q_{del0}$ | 0 | # | $q_{del0}$ | $\leftarrow$ |
| $q_{del0}$ | 1 | # | $q_{del0}$ | $\leftarrow$ |
| $q_{del0}$ | # | 0 | $h$ | |
| $q_{del1}$ | 0 | # | $q_{del1}$ | $\leftarrow$ |
| $q_{del1}$ | 1 | # | $q_{del1}$ | $\leftarrow$ |
| $q_{del1}$ | # | 1 | $h$ | |

**Problem 4.2** Explain in detail what the Halting problem is and its significance. 15pt

10min

# 5 Internet

**Problem 5.1 (IPv4 packet structure.)**

0pt

1. Explain each part of an IPv4 header.
2. Explain how the header checksum is calculated and which protocols use it.
3. Given the following IP header in hexadecimal check if it has a valid checksum.
   3720 0041 0042 2e30 0c9a d339 90e2 2200 0000 0002.