COS 212—Algorithms and Complexity
Matrix chaining, and LCS using dynamic programming.
Practical 2 Term 3 1 August 2016

Due before: 23h59, 7 August 2016

---

(1) `int2bin` integer to binary,and (2) Matrix chaining, (3) LCS with dynamic programming.

Warm up by writing a procedure that prints the binary values of a number. Then implement and try out matrix chaining and LCS. Work inside your name file, in `23practical`.

1. Inside your `23practical` make another directory called `1answer`:
   Write code that reads any positive or negative 32-bit integer and returns its binary value. The code that produces the "binary string" must be in a form of a method that returns an `byte` array that holds one binary digit per element. Negative numbers should use two's complement. Read this up in your hardware book or find out from the Internet. Do this in a directory called `1question` inside your `23practical` directory.

2. Inside your `23practical` make another directory called `2answer`:
   Do the matrix chaining example from the notes. Check that your code produces the shortest matrix product chains for an example with a matrix chain of at least 1000 matrices. Suppose the matrices are called A_i for $i \in [0..n]$ with the dimensions, $A_0 : d_0 \times d_1, A_1 : d_1 \times d_2, \ldots, A_n : d_n \times d_{n+1}$. Your code must generate a array containing all the dimensions $d_0, d_1, \ldots, d_{n+1}$ using random numbers less than 100 and then calculate the best chain. Use $n = 10$ for testing purposes. Also determine how many matrices your code can handle in less than 2 seconds. Your code need not do the actual multiplication of matrices, but it needs to calculate and display only the total number of operations for the calculated parenthesization, e.g, if B is an $n \times m$ matrix, and C is an $m \times p$ matrix, and D is a $p \times q$ matrix then the number of ops for $(B \times C) \times D = (n \times m \times p) + n \times p \times q$.

3. Inside your `23practical` make another directory called `3answer`:
   Develop code to apply the LCS dynamic programming algorithm to find the *longest common subsequence* of two strings and their *edit distance*. The edit distance between to subsequences is the total number of edits, i.e., deletions, insertions, and replacements, made. When there is a match between the two strings the edit distance remains the same.

   (a) Read data from two files and use dynamic programming to find their longest common subsequence.

   (b) We have supplied DNA data in two files called `xDNA.txt` and `yDNA.txt`. The files may be found in:
   `/export/home/notes/ds/`. *Do not put these files into your own directory*, but use them by reading them directly without copying.

   (c) Determine what the longest common subsequence is of these two DNA strings by calculating the LCS first of `xDNA.txt` versus `yDNA.txt` and then verifying that this is correct by calculating the LCS again using `yDNA.txt` versus `xDNA.txt`.

   (d) In both cases print the LCS.

4. Ensure that you have a valid `Makefile` in each directory.

5. Submit using the usual `cd; make submit`.
