CSCI 3110 Assignment 6

Posted: 12.11.2019 **Due: 17.11.2019**

Topics: Implementation

For this assignment, solve 3 of the questions from the Programming Contest. I've sketched solutions to A, B and C below. I can see the submissions on the problem server, but **upload your code to Brightspace as well!**

A. Happy and Unhappy Numbers: We create an array $S[0..1000000]$ to store the numbers' statuses — either HAPPY, UNHAPPY, WAITING or BLANK — and an array $P[0..1000000]$ that we will fill such that $P[i]$ stores the number of happy numbers up to and including i. To start with, we set $S[1]$ to be HAPPY and $P[0] = 0$ and all the other entries of S to BLANK. For i from 1 to 1000000, we set $P[i]$ to be $P[i-1]$ plus 1 if `checkStatus(i)` returns HAPPY and plus 0 otherwise.

The procedure `checkStatus(`i`)` returns HAPPY if $S[i]$ is set to HAPPY, and UNHAPPY if $S[i]$ is set either to UNHAPPY or to WAITING. Otherwise, it sets j equal to the sum of the squares of the digits in i, sets $S[i]$ to WAITING, calls `checkStatus(`j`)`, sets $S[i]$ equal to the returned value, and returns $S[i]$.

We read T and then T times we read A and B and return $P[B] - P[A-1]$.

(Note: this solution works but uses quite a lot of memory; we could reduce the space at the cost of a little speed by storing only $S[1..1000]$ and $P[k]$ for multiples k of 1000, then answering queries by computing S from the start of each query interval to the next multiple of 1000, and from the previous multiple of 1000 to the end of the query interval.)

B. Shortlex: We read a number i and print all the bits in the binary representation of $i+1$ strictly to the right of the first 1. An easy way of doing that is with a recursive function that checks if its argument j is strictly greater than 1 and, if so, calls itself on j right-shifted 1 bit and then prints j mod 2.

C. Flood Modeling: We read N and M and create two $(N+2) \times (M+2)$ arrays $H[0..N+1][0..M+1]$ and $W[0..N+1][0..M+1]$ of integers, one to hold the heights of the terrain and the other to hold the maximum heights the water reaches.

We read the terrain heights into $H[1..N][1..M]$ and fill $W[1..N][1..M]$ with copies of -1 (meaning "undefined"). We set the entries in the border around $H[1..N][1..M]$ to -1 and those in the border around $W[1..N][1..M]$ to 0.

We sort the pairs $(1,1), \ldots, (N,M)$ such that (x_i, y_i) precedes (x_j, y_j) if $H[x_i][y_i] < H[x_2][y_2]$, and store them in an array $P[0..NM-1]$. We run through P and, for each pair (x, y), we call `fill(`$x, y, H[x][y]$`)` if $W[x'][y'] \neq -1$ for any $(x', y') \in \{(x-1, y), (x, y-1), (x+1, y), (x, y+1)\}$.

The procedure `fill(`x, y, h`)` first sets $W[x][y]$ to h and then calls `fill(`x', y', h`)` if $H[x'][y'] \leq h$ and $W[x'][y'] = -1$ for any $(x', y') \in \{(x-1, y), (x, y-1), (x+1, y), (x, y+1)\}$.

We print an $N \times M$ grid filled with $W[1][1] - H[1][1], \ldots, W[N][M] - H[N][M]$.

Note that some inputs may overflow normal `int` integers, so you should consider `long int` or `long long int` integers. If you want to right-shift a number, it's best to have it `unsigned`.