Lecture 18

More practice problems! This time one of the problems is taken from the 2011 International Olympiad for Informatics (`http://www.ioi2011.or.th/`).

Garden (from the 2011 International Olympiad for Informatics):

You're in a garden with many fountains. Some fountains are connected by trails, and each trail `e` has a beauty factor. No two trails have the same beauty factor. You want to take a path consisting of exactly k trails that ends at some particular fountain. You're allowed to start at any fountain you wish. From wherever you start, you first take the most beautiful trail. From then on, you take the most beautiful trail from your current location, unless it's the trail you just came on, in which case you take the 2nd most beautiful trail; if there is no 2nd trail, you just go back on the same trail you came from.

Given the movement rule above, how many ways are there to take a path of length exactly k ending at the target fountain?

Example solution: The most obvious way to solve this problem is to just loop over every possible starting vertex, follow the path of length k according to the rule above, and see where you end up. If you end up at the target fountain, add 1 to a counter, then return the counter at the end. The running time is $\Theta(nk+m)$ if for each edge you first calculate the most beautiful and second most beautiful trail leaving it.

A better solution takes time only $\Theta(n\log_2 k+m)$. The idea is as follows: suppose that for each vertex v we know where we would end up starting from v after taking $1, 2, 4, 8, \ldots$ steps (for all powers of 2 up to k). Then, for each vertex, rather than simulate the enter length-k path, we can skip and take 2^j steps at once for the largest $j = \lfloor \log_2 k \rfloor$. Then, from the vertex we wind up at, there may be more steps to take ($k - 2^{\lfloor \log_2 k \rfloor}$ steps), so we take the next biggest power of 2 number of steps we can, etc., until we finally have taken a total of k steps. Thus, figuring out where a path of length k ends only takes $\lceil \log_2 k \rceil$ steps instead of k.

To actually implement the above solution, we just need to know four values: $f(v,i,0)$, $f(v,i,1)$, $e(v,i,0)$, $e(v,i,1)$. Here, $f(v,i,0)$ tells us what vertex we would end up at starting from v after having taken 2^i steps, and where the first trail we take out of v is the most beautiful trail from v. $e(v,i,0)$ then tells us what the last edge we would take on this path is. Similarly, $f(v,i,1)$ tells us what vertex we would end up at starting from v after having taken 2^i steps, and where the first trail we take out of v is the second most beautiful trail from v, and $e(v,i,1)$ tells us what the last edge we would take in this case is.

We can fill in all the $f(v,i,0), f(v,i,1), e(v,i,0), e(v,i,1)$ values recursively. When $i = 0$, we can just compute the answer by looking at the most beautiful (or second most beautiful) trail out of v. For $i > 0$:

$$f(v,i,j) = \begin{cases} f(f(v,i-1,j),i-1,1) & \text{if } e(v,i-1,j) \text{ is the most beautiful trail leaving } f(v,i-1,j) \\ f(f(v,i-1,j),i-1,0) & \text{otherwise} \end{cases}$$

$$e(v,i,j) = \begin{cases} e(f(v,i-1,j),i-1,1) & \text{if } e(v,i-1,j) \text{ is the most beautiful trail leaving } f(v,i-1,j) \\ e(f(v,i-1,j),i-1,0) & \text{otherwise} \end{cases}$$

Fibonacci sum: Recall that the Fibonacci sequence is $F_0, F_1, \ldots = 1, 1, 3, 5, 8, 13, 21, \ldots$ (other than the first two numbers, every following number is the sum of the previous two). Develop an algorithm to calculate the sum $F_0 + F_1 + \ldots + F_n$.

Example solution: Let $S_n = F_0 + F_1 + \ldots + F_n$. Note

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} F_i \\ F_{i-1} \\ S_{i-1} \end{bmatrix} = \begin{bmatrix} F_i + F_{i-1} \\ F_i \\ S_{i-1} + F_i \end{bmatrix}.$$

If we let A be the matrix on the lefthand side above, then

$$A^n \cdot \begin{bmatrix} F_1 \\ F_0 \\ S_0 \end{bmatrix} = \begin{bmatrix} F_{n+1} \\ F_n \\ S_n \end{bmatrix}.$$

This, we can compute S_n in $O(\log_2 n)$ arithmetic operations via fast powering. One can also prove that $S_n = F_{n+2} - 1$, so another option is to just reuse the code for computing Fibonacci numbers and subtracting one.