9. 24-3 on Page 613

Arbitrage is the use of discrepancies in currency exchange rates to transform one unit of a currency into more than one unit of the same currency. Suppose that we are given n currencies, c_1, c_2, …, c_n and an $n \times n$ table R of exchange rates, such that one unit of currency c_i buys R[i, j] units of currency c_j.

a. Give an efficient algorithm to determine whether or not there exists a sequence of currencies $\langle c_{i_1}, c_{i_2}, ..., c_{i_n} \rangle$ such that

$$R[i_1, i_2] \times R[i_2, i_3] \times ... \times R[i_{k-1}, i_k] \times R[i_k, i_1] > 1$$

Analyze the running time of your algorithm.

We convert an instance of arbitrage into a graph G as follows. We represent countries as vertices. There is an edge between every pair of countries because there is an exchange rate between every pair of currencies. Hence, the graph is complete. Since the graph is complete, if we can find any cycle such that the product of the currency exchange rates is greater than 1.

One possible method to find such a cycle would be to let the edge weights in the graph be the currency exchange rates between the respective countries. Instead, we perform the following transformation, which allows us to solve this problem by determining whether or not the graph contains a negative weight cycle.

$$R[i_1, i_2] \times R[i_2, i_3] \times ... \times R[i_{k-1}, i_k] \times R[i_k, i_1] > 1$$

$$\frac{1}{R[i_1, i_2] \times R[i_2, i_3] \times ... \times R[i_{k-1}, i_k] \times R[i_k, i_1]} < 1$$

$$\frac{1}{R[i_1, i_2]} \times \frac{1}{R[i_2, i_3]} \times ... \times \frac{1}{R[i_{k-1}, i_k]} \times \frac{1}{R[i_k, i_1]} < 1$$

$$\log\left(\frac{1}{R[i_1, i_2]} \times \frac{1}{R[i_2, i_3]} \times ... \times \frac{1}{R[i_{k-1}, i_k]} \times \frac{1}{R[i_k, i_1]} \right) < \log 1 = 0$$

$$\log \frac{1}{R[i_1, i_2]} + \log \frac{1}{R[i_2, i_3]} + ... + \log \frac{1}{R[i_{k-1}, i_k]} + \frac{1}{\log R[i_k, i_1]} < 0$$

Edge weights are defined as follows:

$$w(i, j) = \log\left(\frac{1}{R[i, j]} \right)$$

Now, all we need to do is check graph G for a negative weight cycle. Since the graph is complete, we can run a DFS starting from any vertex. This would require O(V + E) time.

b. Give an efficient algorithm to print out such a sequence if one exists. Analyze the running time of your algorithm.

Run a DFS starting from each vertex. Implement the DFS using a stack. When a negative-weight cycle is detected, we use the stack to reconstruct the cycle.

Let S be a stack. Suppose we have started our DFS from vertex v_1, and that a negative-weight cycle is detected starting/ending at vertex v_i. Then, the contents of the stack at the time of the discovery of the negative weight cycle is as follows:

$S = v_1, v_2, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots v_k$.

We discover the negative weight cycle when examining edge (v_k, v_i). We can print the cycle in reverse as follows:

$v_i, v_k, v_{k-1}, \ldots, v_{i+1}, v_i$

Reversing this cycle yields the correct reverse cycle:

$v_i, v_{i+1}, \ldots, v_{k-1}, v_k, v_i$

Running the DFS from each vertex requires O(V + E) time in the worst case, just like part a. Printing the contents of the stack requires O(E) time in the worst case, when the cycle contains every currency. The total time is thus O(V + 2E) = O(V + E).