Introduction to Algorithms: 6.006
Massachusetts Institute of Technology March 29, 2011
Professors Erik Demaine, Piotr Indyk, and Manolis Kellis Problem Set 5

Problem Set 5

This problem set is divided into two parts: Part A problems are theory questions, and Part B problems are programming tasks.

Both Part A and Part B questions are due **Monday, April 11th** at **11:59PM**.

Solutions should be turned in through the course website. Your solution to Part A should be in PDF format using LaTeX. Your solution to Part B should be a valid Python file which runs from the command line. A template for writing up solutions in LaTeX is available on the course website. Remember, your goal is to communicate. Full credit will be given only to a correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Part A

Problem 5-1. [17 points] **Arbitrage**

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. For example, suppose that 1 U.S. dollar buys 0.82 Euro, 1 Euro buys 129.7 Japanese Yen, 1 Japanese Yen buys 12 Turkish Lira, and one Turkish Lira buys 0.0008 U.S. Dollars. Then, by converting currencies, a trader can start with 1 U.S. dollar and buy $0.82 \times 129.7 \times 12 \times 0.0008 \approx 1.02$ U.S. dollars, thus turning a profit of 2 percent.

Suppose that we are given n currencies $c_1, c_2, \ldots, c_n$ and an $n \times n$ table of R exchange rates, such that one unit of currency c_i buys $R[i, j]$ units of currency c_j. Give an efficient algorithm to determine whether there exists a sequence of currencies $\langle c_{i_1}, c_{i_2}, \ldots, c_{i_n} \rangle$ such that

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

Analyze the running time of your algorithm.

Solution: We can use the Bellman-Ford algorithm on a suitable weighted, directed graph $G = (V, E)$, which we form as follows. There is one vertex in V for each currency, and for each pair of currencies c_i and c_j, there are directed edges (v_i, v_j) and (v_j, v_i). (Thus, $|V| = n$ and $|E| = n(n-1)$.) To determine edge weights, we start by observing that

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

if and only if

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

Taking logs of both sides of the inequality above, we express this condition as

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

Threfore, if we define the weight of edge (v_i, v_j) as

$$\begin{aligned} w(v_i, v_j) &= \lg \frac{1}{R[i, j]} \\ &= -\lg R[i, j], \end{aligned}$$

then we want to find whether there exists a negative-weight cycle in G with these edge weights.

We can determine whether there exists a negative-weight cycle in G by adding an extra vertex v_0 with 0-weight edges (v_0, v_i) for all $v_i \in V$, running BELLMAN-FORD from v_0, and using the boolean result of BELLMAN-FORD (which is TRUE if there are no negative-weight cycles and FALSE if there is a negative-weight cycle) to guide our answer. That is, we intvert the boolean result of BELLMAN-FORD.

This method works because adding the new vertex v_0 with 0-weight edges from v_0 to all other vertices cannot introduce any new cycles, yet it ensures that all negative-weight cycles are reachable from v_0.

It takes $\Theta(n^2)$ time to create G, which has $\Theta(n^2)$ edges. Then it takes $O(n^3)$ time to run BELLMAN-FORD. Thus, the total time is $O(n^3)$.

Problem 5-2. [17 points] **Negative Weight Edges**

Let $G = (V, E, w)$ be a weighted directed graph with exactly two negative-weight edges and no negative-weight cycles. Give an algorithm to find the shortest path weights $\delta(s, v)$ from a given $s \in V$ to all other vertices $v \in V$ that has the same running time as Dijkstra.

Solution: Let the two-negative weight edges be (v_1, t_1) and (v_2, t_2). Then the possible paths from s to any vertex u are:

1. $s \rightsquigarrow u$,
2. $s \rightsquigarrow v_1 \to t_1 \rightsquigarrow u$,
3. $s \rightsquigarrow v_2 \to t_2 \rightsquigarrow u$,
4. $s \rightsquigarrow v_1 \to t_1 \rightsquigarrow v_2 \to t_2 \rightsquigarrow u$, and
5. $s \rightsquigarrow v_2 \to t_2 \rightsquigarrow v_1 \to t_1 \rightsquigarrow u$.

Remove the negative weight edges and use Dijkstra to determine shortest paths from s, t_1, and t_2. Then the shortest paths from s to any vertext u is the minimum over:

$d[s, u]$,

$d[s, v_1] + w[v_1, t_1] + d[t_1, u]$,
$d[s, v_1] + w[v_2, t_2] + d[t_2, u]$,
$d[s, v_1] + w[v_1, t_1] + d[t_1, v_2] + w[t_2, v_2] + d[v_2, u]$, and
$d[s, v_1] + w[v_2, t_2] + d[t_2, v_1] + w[t_1, v_1] + d[v_1, u]$.

Problem 5-3. [16 points] **Integer Weights**

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \{0, 1, \ldots, W\}$ for some nonnegative integer W. Modify Dijkstra's algorithm to compute the shortest path weights from a given source vertex $s \in V$ in $O(WV + E)$ time.

Solution: Notice shortest-path is at most $(V-1)W$, and also an integer. Also notice that EXTRACT-MIN calls are monotonically increasing over time in Dijkstra's algorithm. So we can implement a min-priority queue so that for any sequence of m inserts, extract-min, and decrease-key operations take $O(m + k)$ time, where keys are known to be integers in the range 0 to k and key values extracted are monotonically increasing over time. So let $k = (V - 1)W$, our running time is $O(V + E + VW)$.