COMS21103: Problem set 6

2015/2016

Remark: Most problem sets for the next few weeks will contain at least one starred problem, which is more challenging. If any of the problems seem unclear, please post a question on the Blackboard discussion board.

1. Arbitrage is a (real) method that can be used to make money from fluxations in currency exchange rates. The idea is to convert one unit of a currency into more than one unit of the same currency. For example imagine that 1 U.S. dollar buys 0.7 UK pounds, 1 UK pound buys 1.5 euros and 1 euro buys 1.1 dollars. By converting 1 dollar into pounds then into euros then into back dollars, you would end up with 1.155 dollars; This is a profit of slightly more than 15%!

   Consider a system with n currencies $c_1, c_2 \dots c_n$ where one unit of currency c_1 buys $r_{i,j}$ units of currency c_j. Give an efficient algorithm to determine whether arbitrage exists (for the current exchange rates). What is the running time of your algorithm?

   Hint: Consider the fact that $\log(ab) = \log(a) + \log(b)$.

2. Consider the following modification to Bellman-Ford (which is called Yen's improvement). Assume that the vertices are numbered $v_1, v_2, \dots v_{|V|}$ (any ordering is fine). Separate the edges into two sets E_f and E_b. The set E_f contains every edge (v_i, v_j) such that $v_i < v_j$. The set E_b contains every edge (v_i, v_j) such that $v_i > v_j$. Notice that each edge is in exactly one set (assuming we don't have self loops). In each iteration, instead of relaxing the edges in an arbitrary order, we relax them in the following order. First we relax each edge $(v_i, v_j) \in E_f$ in increasing order of i. For two edges (v_i, v_j) and $(v_i, v_{j'})$, the order remains arbitrary. Second we relax each edge $(v_i, v_j) \in E_b$ in decreasing order of i (again breaking ties arbitrarily). Prove that after this modification, Bellman-Ford's algorithm only needs $\lceil |V|/2 \rceil$ iterations instead of $|V|$ iterations. Notice that even though this doesn't change the time complexity it is almost twice as fast!

3. Every day you drive across Bristol from your home (h) to your work (w). You can think of Bristol of a directed graph consisting of nodes (junctions) and edges (roads). The weight of edge (u, v) is length of the road between junction u and junction v. You know that you can find the shortest path to work by running Dijkstra's algorithm.

   To allow for road-blocks you decide to allow for one road being closed on each journey. Being organised you decide to compute the shortest path for each possible road closure in advance. This means that on a given day, you can simply look up the shortest path that avoids the closed road. Give an algorithm for this problem that runs in $O(|V||E| \log |V|)$ time. Formally, your algorithm should produce $|E|$ outputs; For each edge $e \in E$, the you should output the length of the shortest path between h and w that doesn't use edge e. You don't need to work out how to output the paths themselves (though in practice you would want to).