Problem Set 5

This problem set is due **Wednesday, April 18** at **11:59PM**.

Solutions should be turned in through the course website. **You must enter your solutions by modifying the solution template (in Python) which is also available on the course website.** The grading for this problem set will be largely automated, so it is important that you follow the specific directions for answering each question.

For multiple-choice and true/false questions, no explanations are necessary: your grade will be based only on the correctness of your answer. For all other non-programming questions, full credit will be given only to correct solutions which are described clearly and concisely.

Programming questions will be graded on a collection of test cases. Your grade will be based on the number of test cases for which your algorithm outputs a correct answer within time and space bounds which we will impose for the case. Please do not attempt to trick the grading software or otherwise circumvent the assigned task.

---

1. **Graph Transformations (15 points)**

   You are given a directed graph $G = (V, E)$ with positive or negative weights $w(i, j)$ and no negative cycles. Your job is to find a transformation from weights $w(i, j)$ to new weights $w'(i, j)$ that eliminates the negative edges but does not change the sequence of vertices for each shortest path between any two vertices. Call such a transformation "good" if all shortest path vertex sequences in the graph with weights w' are the same as the shortest path vertex sequences in the graph with weights w.

   For each part below, answer whether the transformation is good. That is, answer `True` if the transformation is good, and `False` if it is not good.

   (a) Replace each weight with its square so that $w'(i, j) = w^2(i, j)$.

   (b) Add a large constant C to each edge weight, so that the weights $w'(i, j) = w(i, j) + C$ all become nonnegative.

   (c) Suppose it is possible to find a value $d(v)$ assigned to each vertex v of the graph such that $w(i, j)+d(i)-d(j) \geq 0$ for each edge (i, j).[1] Take $w'(i, j) = w(i, j)+d(i)-d(j)$.

[1] It is possible to compute such d values by using a variant of the Bellman-Ford algorithm: Make a new source vertex s, connect s to every vertex by a weight-0 edge, run Bellman-Ford starting from s, and let $d(v)$ be the length of the shortest path from s to v.

2. **Topological Sort (25 points)**

Consider the DFS code for directed graphs from CLRS. (This code iterates through all vertices in the graph, and runs DFS starting from this vertex if the vertex has not yet been visited in a prior search.) One can use this DFS to obtaining a topological sort of a directed acyclic graph (DAG) G. (In a topological sort, your goal is to obtain an ordering of the vertices such that all directed edges go from a vertex to a vertex later in the ordering.)

For each of the below proposals, answer `True` or `False` to the following: Running the algorithm on a DAG necessarily produces a topological sorting.

(a) Run the DFS code from CLRS and order the vertices in increasing order of their start time.

(b) Run the DFS code from CLRS, where we start DFS only from sources (vertices with no incoming edges) and sort in increasing order of the start time.

(c) Run the DFS code from CLRS on the reverse of the graph (where we reverse the direction of all directed edges), and where we start DFS only from sources of the reversed graph (vertices with no incoming edges), and sort in decreasing order of the start time.

(d) Run the DFS code from CLRS, and order vertices in decreasing order of their finishing time.

(e) Run the DFS code from CLRS on the reverse of the graph, and order vertices in increasing order of their finishing time.

3. **Making Unlimited Money (40 points)**

You decide to use your MIT education play the stock market. Being an ambitious 6.006 student, you desire not just to make large amounts of money, but to make *unlimited money* through a sequence of financial transactions. We will model this problem by a walk on a directed graph, where each node represents a state of the stock market. If there is a directed edge (i, j) in the graph, then it is possible, by making some financial decision, to move from state i to state j.

Each edge (i, j) has an associated nonnegative real number value, denoted $m(i, j)$, representing the *multiplicative* change in your total cash assets as you move from i to j. (If you have d dollars at state i and take the edge (i, j), you will have $d \cdot m(i, j)$ dollars in state j. Thus, a m value greater than 1 denotes an increase in money, while a value less than 1 denotes a decrease in money.)

Your goal is to design an efficient algorithm to determine if it is possible, starting with 1 dollar and beginning from a start vertex s, to obtain arbitrarily large amounts of money by making a series of financial transactions. At no intermediate step is your cash balance allowed to be below some threshold b (with $b > 0$), since if you do so your broker will not allow you to play the market further.

Formally, your task is to determine (yes/no) whether the graph has the following property:

For any positive integer N, there is a sequence of steps beginning from s with 1 dollar such that you have at least N dollars at the end of the sequence and at no point in the sequence did your balance become less than b.

Design an efficient algorithm for this problem, argue its correctness, and explicitly state its asymptotic running time in terms of $|V|$ (the number of states in the graph) and/or $|E|$ (the number of edges).

4. **Shortest paths on expanders, in sub-linear time (30 points)**

   Suppose we construct an undirected graph in the following way: Fix some small even value d, and for each of the n vertices, choose $\frac{d}{2}$ random neighbors and create those edges. Such a graph is an example of an **expander graph**, with expansion d. For a graph like this, the number of nodes within distance k of a node is roughly d^k, for $k < \frac{\log(n)}{2\log(d)}$ (i.e. when the square of the neighborhood size, d^{2k}, is less than the number of nodes, n). For example, for an expander with expansion factor $d = 10$ and $n = 10^{100}$, a node will have about 10 neighbors, 100 nodes within distance 2, and 10 billion nodes within distance 10.

   We've seen how to use breadth-first search starting from s to find the shortest path from s to t on an undirected (unweighted) graph. But in the special case of expanders, one can actually do much better than $\Theta(E) = \Theta(nd)$ in the average case. Your job will be to design and code a function `find_distance(graph,s,t)` which quickly returns the shortest path from s to t on an expander, or `None` if there is no path (though this is extremely unlikely to happen if $d > 1$).

   Your code should pass the following test case (where $d = 2$):

```
graph = {1: [4, 5],
         2: [3, 4, 5, 6],
         3: [2, 6],
         4: [1, 2, 5, 6],
         5: [1, 2, 4],
         6: [2, 3, 4]}

find_distance(graph, 1, 1) == 0
find_distance(graph, 1, 2) == 2
find_distance(graph, 1, 3) == 3
find_distance(graph, 3, 4) == 2
find_distance(graph, 4, 5) == 1
find_distance(graph, 5, 6) == 2
```