Problem 1

(Help Brito Get Rich!) Professor Brito has a plan (totally hypothetical, of course) to fund his next vacation to Cuba, and he needs your help! He has discovered a clever money-making scheme that works as follows:

Suppose 1 US dollar buys 110 yen, 1 yen buys 0.01 euros, and 1 euro buys 1.1 dollars. Then, a trader starting with \$1 can convert their money from dollars to yen, then from yen to euros, and finally from euros back to dollars, ending with \$1.21! This cycle of currencies $\$ \to ¥ \to € \to \$$ is called an **arbitrage cycle**. However, finding and exploiting these arbitrage cycles before the prices are corrected requires extremely fast algorithms.

In a more general setting, let's say n different currencies are traded in some financial market. Public information gives us the matrix $\text{Er}[1 \ldots n, 1 \ldots n]$ of non-negative exchange rates between every pair of currencies; for each i and j, one unit of currency i can be traded for $\text{Er}[i, j]$ units of currency j. (Do not assume that $\text{Er}[i, j] \cdot \text{Er}[j, i] = 1$.)

a) Describe an algorithm that returns an array $\text{MaxAmt}[1 \ldots n]$, where $\text{MaxAmt}[i]$ is the maximum amount of currency i that you can obtain by trading, starting with one unit of currency 1, assuming there are <u>no</u> arbitrage cycles. **We expect:** a clear graph setting to solve this problem, an algorithm that yields the solution, and the runtime of your algorithm.

b) Describe an algorithm to determine whether the given matrix of currency exchange rates creates an arbitrage cycle. **We expect:** a clear graph setting to solve this problem, an algorithm from class that yields the solution, and the runtime of your algorithm.

c) Modify your algorithm from part (b) to actually return an arbitrage cycle, if it exists. **We expect:** the description of your modification to the algorithm used in part (b), a brief explanation explaining why your algorithm is correct, and the runtime of your design.

Problem 2

(Help Brito Stay Rich!) Thanks to your advice, Professor Brito is now a tycoon of industry. With the profits from his arbitrage strategies, he has diversified his investments. His wealth is now scattered across different banks in various countries around the world to efficiently exploit arbitrage opportunities. However, to fully capitalize on this, he needs to ensure that he can swiftly and profitably transfer assets between these banks.

Picture the global network of banks as a directed graph with weighted edges $G = (V, E, w)$. Here, w_e indicates the cost (or profit) of transferring between two banks; these weights can be positive, negative, or zero, depending on the situation. The banks (vertices) are partitioned into k disjoint clusters, representing different countries: $V_1, V_2, \ldots, V_k$. Each cluster encompasses all the banks in a specific country.

For strategic asset management, Professor Brito is interested in understanding the best possible transfer rate between any two clusters. That is, when he moves his assets from a bank in one country to another, what's the best rate he can secure? Formally, for every pair of clusters V_i and V_j, he wishes to determine the optimal transfer rate, given by

$$\min_{v_i \in V_i, v_j \in V_j} \operatorname{dist}(v_i, v_j).$$

As his trusted financial consultant, your task is to design an efficient algorithm that calculates the optimal transfer rate for all pairs of clusters i and j. For full credit, your algorithm must run in time $\mathcal{O}(k|E|\log(|V|) + |V||E|)$[1]. **We expect:** a description of your algorithm, a brief explanation of its correctness, and analysis of its runtime.

Here, the dist() function is defined as the length of the shortest path from u to v, with a few boundary cases:

1. If there is no path from u to v, then there is no shortest path from u to v; in this case, we define dist$(u, v) = \infty$.
2. If there is a negative cycle between u and v, then there are paths from u to v with arbitrarily negative length; in this case, we define dist$(u, v) = -\infty$.
3. Finally, if u does not lie on a negative cycle, then the shortest path from u to itself has no edges, and therefore doesn't have a last edge; in this case, we define dist$(u, u) = 0$.

We did not discuss too deeply about the time complexities of the shortest path algorithms: On this homework (and future work) assume the following complexities.

- Dijkstra's Algorithm runs in time $\mathcal{O}(|E|\log(|V|))$.
- Bellman-Ford runs in time $\mathcal{O}(|E||V|)$.
- Floyd-Warshall runs in time $\mathcal{O}(|V|^3)$.

[1] $\mathcal{O}(k|V|\log(|V|) + |V||E|)$ is fine as well.

Problem 3

(Matchings...again) Trees are special case of bipartite graphs (if you have no cycle, you have no cycle of odd length). In this problem you will use Dynamic Programming to find a matching of maximum size. Your answer should include:

(1) A description of your DP states, in plain English, including the dimensions.

(2) A mathematical recurrence relation between sub-states. Briefly explain why your recurrence yields the correct answer.

(3) The base case(s) that would allow you to compute the DP states.

(4) How do you get the final answer from the DP states you defined in part (1).

(5) State the runtime of your design (in big-$\mathcal{O}$ notation) and briefly justify your answer.

Hint: it may be useful to track two numbers, depending on whether the root of your subtree is matched or not.

Appendix

Problem 1 is modified from Jeff Erickson's Instroduction to Algorithms, but is also a fairly common problem.