1. Given an undirected graph $G = (V, E)$ with three special vertices u, v, and w, describe and analyze an algorithm to determine whether there is a simple path from u to w that passes through v.

   [Hint: Do **not** *try to modify a standard graph traversal algorithm like DFS or BFS. This problem is on this homework for a reason.]*

Solution (disjoint paths): We construct a new graph G' from G by adding one new vertex t and two new edges ut and wt. We compute the maximum number of vertex-disjoint paths between v and t, as described in the textbook. Finally, we return TRUE if we find exactly two vertex-disjoint paths, and FALSE otherwise.

The analysis in the textbook implies that our algorithm runs in $O(VE)$ time, but we can improve this time bound. The disjoint-paths algorithm ultimately reduces to computing a maximum flow in a related directed graph H with integer edge capacities. In particular, the number of paths is equal to the value of this maximum flow. Because there are at most 2 vertex-disjoint paths in G', the value of the maximum flow in H is at most 2. It follows that Ford-Fulkerson needs only two iterations to compute the maximum flow in H. We conclude that our algorithm actually runs in ***$O(E)$ time***.

Proof of correctness: Because t has only two incident edges in G', there are *at most* two vertex-disjoint paths from v to t in G'. If there are two vertex-disjoint paths, concatenating them and removing the edges ut and wt yields a simple path between u and w that passes through v. On the other hand, is there is a simple path between u and w that contains v, then adding the edges ut and wt to this path creates a simple cycle through v and t, which can be decomposed into two disjoint paths between v and t. ■

Solution (maximum flows): First we construct a new directed graph $G' = (V', E')$ as follows:

- V' contains two vertices v_{in} and v_{out} for each vertex $v \in V$, plus a new target vertex t.
- E' contains three types of edges:
  - E' contains one edge $x_{\text{in}} \to x_{\text{out}}$, with capacity 1, for each vertex $x \in V$.
  - E' contains two directed edges $x_{\text{out}} \to y_{\text{in}}$ and $y_{\text{out}} \to x_{\text{in}}$, each with capacity 1, for each edge $xy \in E$.
  - Finally, E' contains edges $u_{\text{out}} \to t$ and $w_{\text{out}} \to t$, each with capacity 1.

Then we compute a maximum flow from v_{out} to t in G'. If the value of this maximum flow is 2, we return TRUE; otherwise, we return FALSE. The maximum flow value is *at most* 2. Thus, if we compute the maximum flow using off-the-shelf Ford-Fulkerson, our algorithm runs in $O(E' \cdot |f^*|) = O(E') =$ ***$O(V + E)$ time***. ■

Rubric: 10 points: standard reduction rubric. No penalty for implicitly assuming that the input graph is connected, by claiming the algorithm runs in $O(E)$ time. Max 8 points for $O(VE)$ time; scale partial credit.

2. (a) Prove that a directed graph G with no isolated vertices is Eulerian if and only if (1) G is strongly connected and (2) the in-degree of each vertex of G is equal to its out-degree. *[Hint: Flow decomposition!]*

Solution: First, suppose G is Eulerian. Let W be a closed walk in G that traverses each edge exactly once. Because G has no isolated vertices, W visits every vertex of G. In particular, for any two vertices u and v, we can follow W from u to v, and symmetrically, we can follow W from v back to u. Thus, G is strongly connected. For each vertex v, the walk W must enter v and leave v the same number of times; thus, the in-degree and out-degree of v are equal.

On the other hand, suppose G is strongly connected and has balanced degrees. Consider the function $f : E \to \mathbb{R}$ that assigns the value 1 to every edge. Because every vertex has equal in- and out-degrees, f is a circulation in G. The flow decomposition theorem implies that f is the sum of a finite set of edge-disjoint cycles $C_1, C_2, \ldots$.

We can systematically merge the cycles $C_1, C_2, \ldots$ into a single closed walk as follows. We maintain a single closed walk W, which is initially the cycle C_1, and an arbitrary starting vertex s in C_1. Then as long as W does not traverse every edge of G, repeat the following:

- Let $x \to y$ be any edge that is *not* traversed by W.
- Let π be any path from s to y that ends with the edge $x \to y$; this path must exist because G is strongly connected.
- Let $p \to q$ be the *first* edge of π that is not traversed by W. Our closed walk W *must* visit p, since otherwise, we could have chosen an earlier edge of π. We could combine these three steps by letting $p \to q$ be any edge that is not traversed by W, but whose tail vertex p is visited by W. But then we'd have to prove that such an edge exists!
- Let C_i be the decomposition cycle that contains edge $p \to q$.
- Let W' be the closed walk that starts at p, walks all the way around W back to p, and then walks all the way around C_i (starting with the edge $p \to q$) back to p.
- Finally, set $W \leftarrow W'$.

Each iteration of this loop adds at least one vertex to W, so the loop eventually terminates, at which point W is an Euler tour of G. ■

Rubric: 10 points = 5 for each direction. A full-credit solution must include an **algorithm** to merge the decomposition cycles into a single closed walk. But we don't need a running time, because the problem only asks for an existence proof.

(b) Suppose that we are given a strongly connected directed graph G with no isolated vertices that is *not* Eulerian, and we want to make G Eulerian by duplicating existing edges. Each edge e has a duplication cost $€(e) \geq 0$. We are allowed to add as many copies of an existing edge e as we like, but we must pay $€(e)$ for each new copy. On the other hand, if G does not already have an edge from vertex u to vertex v, we cannot add a new edge from u to v.

Describe an algorithm that computes the minimum-cost set of edge-duplications that makes G Eulerian.

Solution: Let G be the given input graph. Assign the lower bound $\ell(e) = 1$, infinite capacity $c(e) = \infty$, and cost $€(e)$ to each edge e. Let ϕ be a minimum-cost feasible circulation in G. Because all the lower bounds are integral, we can assume that the circulation ϕ is also integral.

Let Φ be the graph with the same vertices as G but with $\phi(e)$ copies of each edge e of G. We can interpret ϕ as a circulation that has value 1 on every edge of Φ. Following the algorithm in part (a), we can decompose ϕ into a set of cycles in Φ, which we can then recombine into a an Euler tour of Φ.

The duplication cost to transform G into Φ is $\sum_e (\phi(e) - 1) \cdot €(e)$, which is exactly $\sum_e €(e)$ more than the cost of the circulation ϕ. Thus, minimizing the cost of ϕ is equivalent to minimizing the cost of duplicating edges to get from G to Φ.

Suppose we initialize ϕ by setting $\phi(e) = 1$ for every edge e (to get rid of the positive lower bounds) and then use the successive shortest-path algorithm to compute a minimum-cost feasible flow in the residual graph G_ϕ. The initial residual graph has a total imbalance of at most $O(E)$, so the successive shortest path algorithm ends after $O(E)$ iterations. Thus, our overall algorithm runs in ***$O(E^2 \log V)$ time.*** ■

Rubric: 5 points **extra credit**, standard graph reduction rubric (scaled). This is not the only correct solution. No penalty for using a slower polynomial-time min-cost flow algorithm.

3. (a) Describe and analyze an efficient algorithm that either rounds a given $m \times n$ matrix A, or correctly reports that no such rounding is possible.

Solution: Assume without loss of generality that every row and every column of A sums to an integer, because no legal rounding is possible otherwise. To simplify the problem, we write A as the sum of two $m \times n$ arrays I ("integer") and F ("fractional") by setting

$$I[i,j] := \lfloor A[i,j] \rfloor \quad \text{and} \quad F[i,j] = A[i,j] - I[i,j]$$

for each i and j. If F' is a legal rounding for F, then $I + F'$ is a legal rounding for A. So we only need to compute a legal rounding of the fractional matrix F. For example:

$$\begin{bmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 2 \\ 3 & 4 & 2 \\ 7 & 1 & 0 \end{bmatrix} + \begin{bmatrix} \mathbf{.2} & \mathbf{.4} & \mathbf{.4} \\ \mathbf{.9} & \mathbf{.0} & \mathbf{.1} \\ \mathbf{.9} & \mathbf{.6} & \mathbf{.5} \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 3 & 4 & 2 \\ 7 & 1 & 0 \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} \end{bmatrix} = \begin{bmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{bmatrix}$$

To find a legal rounding for the fractional matrix F, we construct a flow network G with the following vertices and edges:

- A source s, a vertex r_i for each row i, a vertex c_j for each column j, and a target t;
- An edge $s \to r_i$ with capacity $\sum_k F[i,k]$ for each row i;
- An edge $c_j \to t$ with capacity $\sum_k F[k,j]$ for each column j;
- An edge $r_i \to c_j$ with capacity 1 for each row i and column j such that $F[i,j] > 0$.

Next we compute a maximum flow f^* in G from s to t. Because every edge capacity is an integer, we can assume without loss of generality that f^* is an *integer* flow; in particular, every flow value $f^*(r_i \to c_j)$ is either 0 or 1. Finally, for each row i and column j, set $F^*[i,j] \leftarrow f^*(r_i \to c_j)$.

Correctness: I claim that F^* is a legal rounding of F if and only if the maximum flow f^* saturates every edge leaving s (and therefore saturates every edge entering t). We prove each implication separately:

($\Leftarrow$) Suppose f^* saturates every edge leaving s and every edge entering t. Then for each row index i, we have

$$\begin{aligned} \sum_k F^*[i,k] &= \sum_k f^*(r_i \to c_j) && \text{[definition of } F^*[i,k]] \\ &= f^*(s \to r_i) && \text{[flow conservation at } r_i] \\ &= c(s \to r_i) && \text{[definition of saturated]} \end{aligned}$$

$$= \sum_k F[i,k] \qquad \text{[definition of } c(s\to r_i)\text{]}$$

In short, every row of F^* has the same sum as the corresponding row of F. A symmetric argument implies that each column of F^* has the same sum as the corresponding column of F. Finally, every entry $F^*[i,j]$ is either 0 or 1. We conclude that F^* is a legal rounding of F.

($\Rightarrow$) On the other hand, suppose F^* is a legal rounding of F. Then for each row index i, we have

$$\begin{aligned} f^*(s\to r_i) &= \sum_k f^*(r_i\to c_j) && \text{[conservation at } r_i\text{]} \\ &= \sum_k F^*[i,k] && \text{[definition of } F^*[i,k]\text{]} \\ &= \sum_k F[i,k] && \text{[definition of rounding]} \\ &= c(s\to r_i) && \text{[definition of } c(s\to r_i)\text{]} \end{aligned}$$

We conclude that f^* saturates every edge leaving s. A symmetric argument implies that f^* saturates every edge entering t.

Running time: Our network has $O(m+n)$ vertices and $O(mn)$ edges. Thus, if we compute the maximum flow using Orlin's algorithm, our algorithm runs in $O(VE) = \boldsymbol{O(mn(m+n))}$ ***time***. Alternatively, the value of the maximum flow is at most $O(mn)$, so Ford-Fulkerson finds the maximum flow in $O(|f^*|\cdot E) = O(m^2n^2)$ time. ■

Rubric: 7 points: standard graph-reduction rubric (scaled-ish). The running time must be reported as a function of the input parameters n and m.

(b) Prove that a legal rounding is possible *if and only if* the sum of entries in each row is an integer, and the sum of entries in each column is an integer. In other words, prove that either your algorithm from part (a) returns a legal rounding, or a legal rounding is *obviously* impossible.

Solution: One direction is trivial: If any row or column has a non-integer sum, then there is no legal rounding. So suppose each row sum and column sum is an integer.

Consider the non-integral flow f defined by setting $f(r_i \to c_j) = F[i,j]$ for every row i and column j, and saturating every edge $s \to r_i$ and $c_j \to t$. Straightforward definition-chasing implies that f is indeed a flow with value $\sum_{i,j} F[i,j]$. Moreover, f is actually a *maximum* flow, because it saturates every edge leaving s.

It follows that any *integral* maximum flow f^* also has value $\sum_{i,j} F[i,j]$, because all maximum flows have the same value. Thus, f^* also saturates every edge leaving s and every edge entering t. Integrality implies that every flow value $f^*(r_i \to c_j)$ is either 0 or 1. Because f^* saturates every edge out of s, each row of F^* has the same sum as the corresponding row in F; similarly, because f^* saturates every edge into T, each column of F^* has the correct sum. We conclude that F^* is a legal rounding of F. ■

Rubric: 3 points = 1 for the easier direction + 2 for the harder direction.