Solution Set 4

Problem 1 *Let D be a directed graph. Let W be a walk (not an Eulerian walk, just any walk) in the graph. Let v_0 be the last vertex of W. Let T be subgraph of D where $u \to v$ is in T if $u \to v$ is the last edge used to depart from u.*

(a) Show that T has at most one cycle and, if it has a cycle, that v_0 is on this cycle.

Note that every vertex of T has out-degree ≤ 1, so any cycle must be a directed cycle.

Let $v_1 \to v_2 \to \cdots \to v_r \to v_1$ be a cycle of T. Suppose, for the sake of contradiction, that v_0 is not on this cycle. Among v_1, v_2, ..., v_r, let v_i be the vertex our walk visited most recently. But then the last time we visited v_i we left it to go to v_{i+1}, so v_{i+1} was visited later than v_i, a contradiction.

(b) Suppose that G contains an edge $v_0 \to v_1$. Let W' be the walk obtained by concatenating $v_0 \to v_1$ to W and let T' be the graph for W'. Describe T' in terms of W and $v_0 \to v_1$.

If T has an edge from v_0 to some vertex other than v_1, delete that edge. Then add an edge from v_0 to v_1.

(c) Let $\tau(D, v)$ be the number of spanning trees of D rooted at v. Let $\sigma(D, v)$ be the number of connected spanning subgraphs of D with one directed cycle γ, with γ passing through v. Give a simple relation between $\tau(D, v)$ and $\sigma(D, v)$.

We have $\sigma(D, v) = \text{outdeg}(v)\tau(D, v)$. Map the set of graphs with one cycle through v to the set of trees rooted at v by deleting the edge out of v. I claim every fiber of this map has size $\text{outdeg}(v)$. Specifically, given a tree Γ rooted at v, we can expand it to a graph with one cycle through v by adding any one of the $\text{outdeg}(v)$ edges out of v.

Problem 2 Consider the following problem: Given a graph G, with n vertices, determine whether or not it is bipartite. Describe an algorithm to do this and work through a basic estimate of how many steps this will take. The exact answer will depend on your algorithm and on precisely how you model computation. But if your answer is worse than polynomial in n, you are definitely doing something wrong.

I'll assume our graph is given to us as an adjacency matrix. Suppose there are n vertices and e edges. This solution assumes that following links from one vertex to another takes constant time. In parentheses, I'll write out a second solution which actually thinks from a Turing machine perspective, although there isn't a lot of benefit in doing this. I am also not trying to be as efficient as possible here, just to show how an analysis can be done.

We'll have an array of length n which keeps track of whether each vertex is white, black, or not yet colored. (In terms of Turing machines, imagine n dedicated places on the tape, each with one of 3 symbols.)

1. Start out by marking all the vertices as uncolored. Time: linear in n.

2. Go through the list of vertices looking for a vertex which is colored but has an uncolored neighbor. This involves at most n^2 checks of whether u is colored, v is uncolored, and there is an edge from u to v. (Longer on a Turing machine because, for each check, you need to travel a distance of roughly n^2 to get between the region of the tape where the adjacency matrix is and the region where the coloring is. So n^4 on a Turing machine, done in the obvious way.) If all the vertices are colored, output BIPARTITE.

3. If there is no such vertex, choose an uncolored vertex and color it. Time n to find an uncolored vertex. Return to 2.

4. If u is colored and v is uncolored, color v opposite to u. Then check whether v is compatible with all its other neighbors. This is at most $\deg(v)$ tests. In many models of computation, this test takes constant time. (On a Turing machine, each check requires us to travel distance n^2 to get to the relevant part of the adjacency matrix. Actually doing it in this time on a Turing machine is tricky. I think the following works: Put a temporary marker in the coloring array marking which

vertex u you are currently checking as a neighbor of v, and put a similar marker in the adjacency matrix at position (v, u). By marker, I mean add some extra states to the alphabet which encode "white, marker here", and so forth, so we can move the marker without losing the coloring data. Zip back and forth between the two arrays, moving each marker forward one spot, and checking whether we have found a contradiction yet.)

5. If the check in 4 fails, output NOT BIPARTITE.
6. Return to 2.

Problem 3 *Let G be a graph where every vertex has even degree. An* Eulerian orientation *of G is a way to direct the edges of G so that every vertex has in-degree equal to out-degree.*

(a) Show that G has an Eulerian orientation.

Let H be a bipartite graph with equally many black and white vertices. A **prefect matching** *of H is a collection of edges which covers every vertex of H exactly once.*

I did more of this in class than I mean to. Let G_1, G_2, ..., G_r be the connected components of G. Since each G_i has all vertices of even degree, each has an Eulerian tour, and each G_i has an Eulerian orientation. Take these orientations on each component to orient G.

(b) Let G have vertices of degrees $2d_1$, $2d_2$, ..., $2d_m$. Describe a (polynomial time) algorithm which constructs a bipartite graph H with $2\sum d_i$ vertices so that

$$\#\,(\text{perfect matchings of } H) = \#\,(\text{Eulerian orientations of } G) \cdot \prod_i (d_i)!$$

For each vertex v of degree $d(v)$, create $d(v)$ white vertices $W(v,1)$, $W(v,2)$, ..., $W(v,d(v))$. For each edge e (of which there are a total of $\sum d_i$) create a new black vertex $B(e)$. If e joins u and v in G, join $B(e)$ to $W(u,1)$, $W(u,2)$, ..., $W(u,d(u))$, $W(v,1)$, ..., $W(v,d(v))$.

We describe a map from perfect matchings of H to Eulerian orientations of G. Given any perfect matching of H, if $B(e)$ is matched to a vertex of the form $W(u,i)$, orient e toward u. It is easy to check that this gives an Eulerian orientation. For a given Eulerian orientation, in order to lift it to a matching, at every vertex v, we must choose a bijection between the vertices $W(v,1)$, $W(v,2)$, ... and the $d(v)$ edges directed into v; there are $d(v)!$ ways to make such a choice.