CS 280 - Homework 6
Solutions

Problem 1.

a. Girth: 3

b. Circumference: 14

c. Diameter: 5

d. Minimum t for which t-partite: 4

e. See accompanying picture.

f. Radius: 3

g. Center: $\{d, j\}$

h. Periphery: $\{a, f, g, h, l\}$

i. Centroid: d

Problem 2,3. See accompanying scans. (To be posted later.)

Problem 4. (6)

a. We have K_6; choose an arbitrary node v. There are 5 edges incident to this node, and thus there are at least 3 incident edges of the same color. (Assume WLOG that the color is red.) Let the nodes on the other end of these edges be a, b, and c. (That is, the edges (v, a), (v, b), and (v, c) are all red.) There are now two possibilities: either one of the edges (a, b), (a, c), (b, c) is red, or none of them are red. If one of them is red (assume WLOG that it is (a, b)), then we have formed a red triangle (in this case, Δ_{vab}). If none of them are red, however, then they must all be blue — thus we have a blue triangle, Δ_{abc}. In either case, we have a monochromatic triangle, so we have proved that when the edges of K_6 are colored with two colors, there always exists at least 1 monochromatic triangle.

c. If r_i is the number of red edges incident to vertex i, then $b_i = (n - 1 - r_i)$ is the number of *blue* edges incident to i. Note that every *non*monochromatic triangle has exactly two vertices where a red and blue edge meet. (This is easy to see if you actually draw out a triangle.) For each vertex i, the number of pairs of a red and a blue edge is $r_i b_i$; thus

$$\sum_{i=1}^{n} r_i b_i$$

seems to be the number of nonmonochromatic triangles. But remember that each triangle has two such vertices: this is where the $\frac{1}{2}$ comes from. Now simply note that the total number of triangles in a complete graph K_n is $\binom{n}{3}$, and thus the number of monochromatic triangles is

$$\binom{n}{3} - \frac{1}{2} \sum_{i=1}^{n} r_i b_i$$

b. The result for K_7 follows directly from application of this formula.

Problem 5. (4) There are two different approaches to this problem. One solution is to perform a DFS rooted at each node, which results in a list for each node of all other nodes that are reachable from it. We then for each node N check the list of every node in N's list for N's inclusion in it - that is, find all pairs which can reach each other. We initialize each node to be a strongly connected component (with itself), and whenever a connected pair of nodes is found, we join their connected components.

Alternatively (and more efficiently), we can modify our DFS to look for cycles by checking for back edges - those edges extending from the node currently being explored to one of its ancestors in the current search. All members of a cycle will surely be in the same SCC, but also if ever a node appears in more than one cycle, the the associated SCC's need to be merged into a single SCC.

Problem 6. (4) The max-flow/min-cut theorem tells us that the maximal amount of flow between a source and a sink is equal to the capacity of a minimal cut, where a cut is defined as a set of edges such that every directed path between the source and sink must pass through at least one edge of the cut. To calculate edge connectivity, we first construct a new graph Γ' over the same set of nodes, but with each undirected edge in Γ replaced with two directed edges in opposing directions each with capacity 1. We then arbitrarily choose a node to be a source, and for every other node in the graph we run the max flow algorithm with it as a sink. By the max-flow/min-cut theorem, the minimal cut for the source and this sink has capacity equal to the max flow F, and hence the cut contains exactly F directed edges (since each edge has capacity 1). If for each [directed] edge in this cut we were to remove the corresponding [undirected] edge in the original graph Γ, there would be no path from the source to the sink and vice versa and thus the graph would be disconnected. If we take the smallest min cut for the source and any sink, we obtain the edge connectivity. Clearly each run of the max flow algorithm is run on a network with $|V|$ vertices and $2|E|$ edges. We run the algorithm exactly $|V|-1$ times, as described above. Note that we DO NOT need to run the algorithm $|V| \cdot (|V|-1)$ times (i.e. run it "for all pairs" of nodes) because we are guaranteed that our arbitrary source will lie in exactly one graph component once it is disconnected, and there will be at least one node in another component, and thus running max flow with that node as a sink will produce the desired (or an equivalently small) disconnection.

Problem 7. (4)

a. There were 10 maximal matchings.

b. If our matching M on Γ is $\{(v_1, v_1'), (v_2, v_2'), \ldots, (v_n, v_n')\}$, then there is a flow with total flow n. Since there is exactly one edge from the source into each of the nodes in L, the total inflow to each node in L is at most 1. We can take a flow that uses only the edges in M, and thus exactly n nodes in R have an inflow of 1. (All those except the v_i' have an inflow of 0, and thus an outflow of 0 as well.) Since all nodes in R are connected directly to the sink, each contributes its entire inflow, and thus the total flow is just n.

Given a flow with total flow n which uses only edges from L to R which form a matching for Γ, there clearly is a matching with n edges on Γ. Since the total flow is n, n nodes in R must have inflow 1, and thus n edges are used in the flow. Since we assume that these edges form a matching, we know that there exists a matching with n edges. We now show that any flow with total flow n can be transformed into one whose edges form a matching, while preserving the total flow of n. Essentially, if there are two edges which go to the same v_i', we can eliminate one of them; since the total outflow of v_i' must be less than 1, the extra edge coming in contributes nothing to the total flow. Similarly, if two edges come out of the same node v_i, we can eliminate one of them; since the total inflow to v_i is at most 1, the extra edge out contributes nothing.

Although we cannot apply this process incrementally, it is possible to eliminate edges such that we have at most one edge coming out of, and going into, each node. (Note that this means that the edges from L to R form a matching for Γ.) Since this process preserves the total flow, we know that the resulting flow still has total flow n. Thus, by the above result, this means that there is a matching with n edges on Γ.