Spring 2010 **CMSC 858Y: Homework 2** Samir Khuller

Due in class: No due date - complete by Mar 22.

(1) Given an edge weighted graph $G = (V, E)$ find a minimum weight subset of edges E' such that every vertex in G is incident on at least one edge in E'. (Such a set is called a min weight edge cover.) Give a polynomial time algorithm for solving minimum weight edge cover. If you cannot solve it for general graphs, you may solve it for bipartite graphs for partial credit. Give an argument to prove correctness of the algorithm. (You may find the following fact useful: a min weight perfect matching in any weighted graph can be found in polynomial time.)

(2) Given a weighted bipartite graph G, design an efficient algorithm to find a perfect matching that minimizes the weight of the maximum weight edge in the matching. In other words, the weight of a matching $w(M) = \max_{e \in M} w(e)$. The algorithm should compute a matching with minimum weight.

(3) An edge coloring of a graph is a coloring of the edges, so that two edges incident on the same node have different colors. A k coloring is a coloring of the edges that uses at most k colors.

A graph is d-regular if every node has degree d. First prove that every d-regular bipartite graph has a perfect matching. Now show that every d-regular bipartite graph can be colored with d colors.

(4) Show that any bipartite graph can be colored with Δ colors where Δ is the maximum degree. (Extend your solution to problem 3.)

(5) Suppose we have a network flow problem in a directed graph $G = (V, E, s, t)$ where the vertices have capacities. Hence the net flow entering a vertex cannot exceed its capacity $c(v)$. How do we find a maximum flow from s to t? Show how to reduce this to the standard flow problem with edge capacities.

(6) The $b - matching$ problem is the following: given a bipartite graph $B = (V, U, E)$ and a function $b : V \cup U \rightarrow Z^+$, is there a subset $M \subseteq E$ such that for each $v \in V \cup U$, v is incident upon exactly $b(v)$ edges in M? How would one design a strongly polynomial algorithm for this problem? (An algorithm is strongly polynomial if its running time depends only on the input size (i.e., number of vertices or edges) and not on the particular numbers/weights. In this case the running time should not depend on the b values.)