**CPS 483/583 – Homework 2**

**Due:** 5 PM, Friday, 19 February, 2021

* Recall that when we say “given (di)graph *G*”, we mean the adjacency lists of *G* are given to work with.
* For vertices *x* and *y*, [*x,y*] denotes an (undirected) edge between *x* and *y* and (*x,y*) refers to the directed edge *x* → *y*.
* In a weighted graph, *weight/length of a path* is the sum of weights of edges on the path. The distance between vertices *x* and *y* is the weight of a shortest path between *x* and *y*.

1. (20 pts) Given an *arbitrary* connected graph *G*, use the discussion on bipartite graphs along with breadth first search, to present an O(*m*+*n*) time algorithm for the following: if *G* is bipartite, output *bipartite* and also find a proper/valid 2-coloring of *G*: for a vertex *v* record the color assigned to vertex *v* (1 or 2) in *v*.color.

If *G* is not bipartite, output *NOT bipartite* and as a certificate for it, the algorithm should output vertices that form a cycle of odd length in the cyclic order.

You must provide the entire algorithm for the problem.

Problem 1. An edge cover C of a graph G(V, E) is a subset of E such that for all v ∈ V there exists e ∈ C with v ∈ e i.e. an edge set covering all vertices. Let C ∗ be a minimum edge cover, that is |C ∗ | ≤ |C| for all edge covers C of G. Prove that |C ∗ | + |M∗ | ≤ |V | where M∗ is a maximal matching of G. Solution: Let S be the set of vertices not covered by M∗ . Note that S is an independent set. Let C be all edges in M∗ plus edges which connect M∗ to S. This is an edge cover. Then, |C ∗ | ≤ |C| = |M∗ | + |S| = |M∗ | + (|V | − 2|M∗ |) = |V | − |M∗ | and rearrange

1. (20 pts) The problem has to do with the following: given graph *G* and vertex *s*, find a shortest path from *s* to each of the other vertices.

We learnt that breadth first search can solve the problem when every edge has a weight/length of 1.

Theorem 2 (Classification). Given a k-connected graph G, vpck (G) = sχk(G). Proof. Given a k-connected spanning subgraph H of G with chromatic number ℓ, color this subgraph properly with ℓ colors. Then between every pair of vertices in H, there are at least k internally disjoint properly colored paths. Thus, using Fact 3, vpck (G) ≤ vpck (H) = ℓ so vpck (G) ≤ sχk(G). Now let ℓ = vpck (G) and consider an ℓ-coloring of G which is properly kconnected. Let P be the set of all proper paths between pairs of vertices (k paths for each pair of vertices). Then the subgraph H of G induced on all the edges of P spans G, is k-connected and has chromatic number at most ℓ. This means vpck (G) ≥ sχk(G), completing the proof.

(a) Consult above for the definition of distance between vertices in a weighted graph.

Suppose you are given a weighted graph *G* in which weight of every edge is one of 1, 2, 3, 4, or 5, and vertex *s*. Present a small example to show that the simple breadth first search from *s* is *not* guaranteed to find the shortest paths from *s* to other vertices.

Clearly present the graph with vertex *s* labeled, and the outcome of a breadth first search from *s*.

Theorem 2 shows that every statement about vpck is a statement about the chromatic number of a minimally k-connected subgraph. Particularly, if G is minimally k-connected, then vpck (G) = χ(G). When the graph is bipartite, we get the following easy observation. Corollary 3. If G is k-connected and bipartite, then for all t ≤ k, we have vpct (G) = 2. In light of the classification theorem, we immediately get equivalent colored “fan lemma” and “disjoint paths between k-sets” versions of the definition of vertex proper connectivity. Corollary 4. A colored graph G is properly k-connected if and only if for every vertex v and k-set of vertices {u1, u2, . . . , uk}, there exists a set of properly colored paths {P1, P2, . . . , Pk} where Pi goes from v to ui and Pi ∩ Pj = {v} for all i, j. Corollary 5. A colored graph G is properly k-connected if and only if for every 2k-set of vertices {u1, u2, . . . , uk, v1, v2, . . . vk}, there exists a set of properly colored paths {P1, P2, . . . , Pk} where Pi goes from ui to vj for some j and Pi∩Pℓ = ∅ for all i, ℓ. Theorem 2, along with Theorem 1, also gives us the following general upper bound. The sharpness of Theorem 1 and Corollary 3 yield the sharpness of both bounds here.

**Read the following before attempting (b) through (e)**

Suppose we have a weighted graph *G* in which weight of an edge is an integer from 1 ··· 5 (i.e. it is one of 1, 2, 3, 4, or 5).

Consider the following construction to build graph *H* based on graph *G*: corresponding to every vertex of *G*, *H* also has a vertex. We will also introduce additional vertices in *H* as explained below.

Every edge in *H* will have a weight of 1.

Whenever *G* has an edge *x*−*y* with weight *k*, *instead* we will introduce in *H* a path connecting *x* and *y* with *k* edges such that each of the intermediate vertices on the path is new, and also every such vertex has degree exactly 2. Observe that we have to add *k* − 1 such new vertices.

For example, if edge *x* − *y* in *G* had a weight of 4 in *G*, in *H* we will instead have the path *x* − 0 − 0 − 0 − *y* connecting *x* and *y*. Each of the 0 vertices is new and degree of such a vertex in *H* is exactly 2.

Clearly, this means, if edge *x* − *y* has a weight of 1 in *G*, then in *H* we will also have the edge *x* − *y*.

1. Let *G* = (*V,E*) where *V* = {1*,*2*,*3*,*4} and *E* = {[1*,*2]*,*[2*,*3]*,*[3*,*4]*,*[1*,*3]*,*[1*,*4]}. Further, weight of [1*,*2] is 1, weight of [2*,*3] is 3, weight of [3*,*4] is 5, weight of [1*,*3] is 5, and weight of [1*,*4] is 2.

Present *G* (along with weights of edges) and *H* (constructed from *G* as above).

1. What is the distance between vertices 1 and 3 in *G*? What is the distance between vertices 1 and 3 in *H*?
2. Given a graph *G* in which weight of every edge is an integer from 1 ··· 5, describe how breadth first search can still be used to find shortest paths in *G* from *s* to other vertices; use observations from (b) and (c).

You do not have to write pseudocode. If you clearly list the steps, that is fine.

1. Recall that *m* is the number of edges and *n* is the number of vertices in the input graph *G*. Analyze your algorithm from (d) to establish its complexity in terms of *m* and *n*. Explain your analysis and outcome.