CS 473 ✧ Spring 2017

Homework 8

Due Wednesday, April 12, 2017 at 8pm

---

1. Recall that a ***path cover*** of a directed acyclic graph is a collection of directed paths, such that every vertex in G appears in at least one path. We previously saw how to compute *disjoint* path covers (where each vertex lies on *exactly* one path) by reduction to maximum bipartite matching. Your task in this problem is to compute path covers *without* the disjointness constraint.

   (a) Suppose you are given a dag G with a unique source s and a unique sink t. Describe an algorithm to find the smallest path cover of G in which every path starts at s and ends at t.

   (b) Describe an algorithm to find the smallest path cover of an arbitrary dag G, with no additional restrictions on the paths. *[Hint: Use part (a).]*

2. Recall that an ***(s, t)-series-parallel*** graph is an directed acyclic graph with two designated vertices s (the *source*) and t (the *target* or *sink*) and with one of the following structures:

   - **Base case:** A single directed edge from s to t.
   - **Series:** The union of an (s,u)-series-parallel graph and a (u,t)-series-parallel graph that share a common vertex u but no other vertices or edges.
   - **Parallel:** The union of two smaller (s,t)-series-parallel graphs with the same source s and target t, but with no other vertices or edges in common.

   Any series-parallel graph can be represented by a binary *decomposition tree*, whose interior nodes correspond to series compositions and parallel compositions, and whose leaves correspond to individual edges. In a previous homework, we saw how to construct the decomposition tree for any series-parallel graph in $O(V+E)$ time, and then how to compute a maximum (s,t)-flow in $O(V+E)$ time.

   Describe an efficient algorithm to compute a *minimum-cost* maximum flow from s to t in an (s,t)-series-parallel graph G in which every edge has capacity 1 and arbitrary cost. *[Hint: First consider the special case where G has only two vertices but lots of edges.]*

3. Every year, Professor Dumbledore assigns the instructors at Hogwarts to various faculty committees. There are n faculty members and c committees. Each committee member has submitted a list of their *prices* for serving on each committee; each price could be positive, negative, zero, or even infinite. For example, Professor Snape might declare that he would serve on the Student Recruiting Committee for 1000 Galleons, that he would *pay* 10000 Galleons to serve on the Defense Against the Dark Arts Course Revision Committee, and that he would not serve on the Muggle Relations committee for any price.

   Conversely, Dumbledore knows how many instructors are needed for each committee, as well as a list of instructors who would be suitable members for each committee. (For example: "Dark Arts Revision: 5 members, anyone but Snape.") If Dumbledore assigns an instructor to a committee, he must pay that instructor's price from the Hogwarts treasury.

Dumbledore needs to assign instructors to committees so that (1) each committee is full, (3) no instructor is assigned to more than three committees, (2) only suitable and willing instructors are assigned to each committee, and (4) the total cost of the assignment is as small as possible. Describe and analyze an efficient algorithm that either solves Dumbledore's problem, or correctly reports that there is no valid assignment whose total cost is finite.