Fall 2018, CMPSC 465: Exam 4 (practice) solutions.

1. There are two types of professional wrestlers: "babyfaces" ("good guys") and "heels" ("bad guys"). Between any pair of professional wrestlers, there may or may not be a rivalry. Suppose we have n professional wrestlers and we have a list of r pairs of wrestlers for which there are rivalries. Give an $O(n + r)$-time algorithm that determines whether it is possible to designate some of the wrestlers as babyfaces and the remainder as heels such that each rivalry is between a babyface and a heel. If it is possible to perform such a designation, your algorithm should produce it.
   Solution:
   Construct an undirected graph where every wrestler is a vertex and an edge is a rivalry. Observe that we are asked to determine if the graph is bipartite, i.e., whether we can designate some wrestlers (vertices) as babyfaces (set A) and the others as heels (set $B = V \setminus A$), so that there are no rivalries (edges) between wrestlers (vertices) of the same "type" (belonging to the same set). We can use the DFS-based bipartiteness algorithm. The algorithm works as follows: We use two colors, say blue and red, and assign each vertex in the graph either the blue group or the red group when performing a DFS. When doing a DFS, we color alternate levels of the DFS tree with different colors. This means that all the tree edges $\langle u, v\rangle$ are such that $color[u] \neq color[v]$. If we encounter a back edge , for the graph to be bipartite, we must have that $color[u] \neq color[v]$. If this back edge property is violated at any point of execution of the algorithm, then we exit saying a partitioning is not possible. If we visit all vertices and edges and still do not encounter any *monochromatic edges*, then we have a valid partitioning of the vertices.

2. The police department in the city of Computopia has made all streets one-way. The mayor contends that there is still a way to drive legally from any intersection in the city to any other intersection, but the opposition is not convinced. A computer program is needed to determine whether the mayor is right. However, the city elections are coming up soon, and there is just enough time to run a linear-time algorithm.

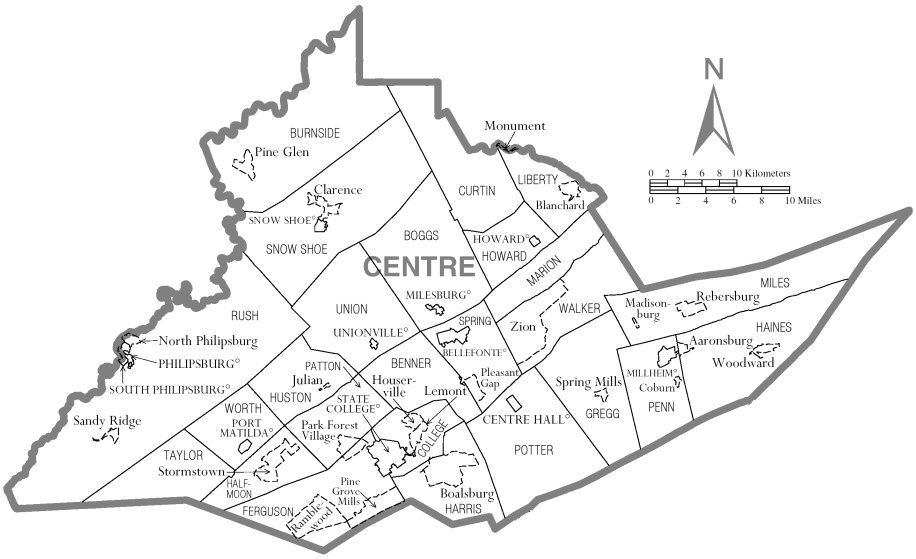
   - Formulate this problem graph-theoretically, and explain why it can indeed be solved in linear time.
     Solution:
     We use a directed graph representation, with vertices representing intersections, and one-way roads denoting directed edges. The mayor's claim is that for any two vertices $u, v \in V$, there is a path from u to v and a path from v to u. This means the mayor is claiming that *all vertices* in the graph belong to a single strongly connected component. This claim can be verified by executing the DFS-based SCC algorithm on the graph (which takes linear time) and checking the number of SCCs produced.
   - Suppose it now turns out that the mayor's original claim is false. She next claims something weaker: if you start driving from town hall, navigating one-way streets, then no matter where you reach, there is always a way to drive legally back to the town hall. Formulate this weaker property as a graph-theoretic problem, and carefully show how it too can be checked in linear time.
     Solution:
     Let town hall be vertex s. Now, the mayor's claim is the following: if v is a vertex reachable from s, then s is reachable from v. In other words, all vertices reachable from s are contained in the SCC that s belongs to. Notice that this can happen only if s belongs to a *sink SCC*. Identifying if an SCC is a sink SCC is a linear-time operation, as we need to create an SCC metagraph. An alternate linear-time strategy is to invoke explore from s and track the visited vertices. If this set of vertices is identical to vertices in s's SCC, then the claim is true.

3. Your job is to arrange n ill-behaved children in a straight line, facing front. You are given a list of m statements of the form "i hates j." If i hates j, then you do not want to put i somewhere behind j, because then i is capable of throwing something at j. Give an algorithm that orders the line, (or says that it is not possible) in $O(m+n)$ time.
Solution:
We will formulate this as a graph problem. Create a directed graph G with each child denoting a vertex. For every "i hates j" relationship, add an edge $\langle i, j\rangle$. *Ordering the line* would mean ordering the vertices. If there is an edge $\langle i, j\rangle$ in G, then i should appear before j in the order. Now, let us consider when an ordering is not possible. One such case is when there are edges $\langle i, j\rangle$ and $\langle j, i\rangle$. In this case, there is clearly no ordering. Similarly, when there is an edge triple $\langle i, j\rangle$, $\langle j, k\rangle$, and $\langle k, i\rangle$, an ordering of vertices is not possible. Thus, we want G to be acyclic or a DAG. If G has a cycle, then no ordering is possible. We can perform a DFS to check if there is a back edge (and a cycle). If we don't find a back edge, then the graph is a DAG. One possible ordering of vertices in a DAG is through a topological sort (specifically, decreasing order of post identifiers when performing a DFS). We can place the vertex appearing first in the topological sort at the head of the line, the second vertex at the second position, and so on. Since both DFS and topological sort are linear time, the overall approach takes $\mathrm{O}(m+n)$ time.

4. Professor Madduri claims that the algorithm for strongly connected components would be simpler if it used the reversed graph in the second depth-first search as well, and scanned the vertices in the order of *increasing* post identifiers. Does this simpler algorithm always produce correct results?
Solution:
No, this algorithm will not work. In the correct algorithm for SCC, we use the property that "the node that receives the highest post number in a depth-first search *must* lie in a *source* strongly connected component." A source component in the reversed graph is a sink component in the actual graph. So we iteratively remove sink components in the actual graph when we do DFS in decreasing post identifier values.

We cannot go by increasing post identifiers on the *reversed graph* itself, because the statement "the node that receives the lowest post number in a depth-first search *must* lie in a *sink* strongly connected component" is incorrect. Here's a small example, a (reversed) graph with the edges $\langle A, B\rangle, \langle B, C\rangle, \langle C, A\rangle, \langle B, D\rangle$. In this case, the increasing post number ordering could be C, D, B, A, and performing a DFS from C will give an incorrect result.

5. The map below shows the townships and boroughs in Centre County, PA. Suppose we want to color this map, so that neighboring townships/boroughs have different colors. We want to determine the minimum number of colors required to do such a coloring. Formulate this "map coloring" problem as a graph-theoretic problem. What is a lower bound on the number of colors required, and a possible upper bound? Briefly explain your answers. Image source:



https://commons.wikimedia.org/wiki/File:Map_of_Centre_County,_Pennsylvania.png

Solution:

Graph-theoretic problem formulation:

Input: an undirected graph $G(V, E)$. Each township/borough is a vertex, and two vertices are connected by an edge iff they share a border.

Output: A color assignment to every vertex $v \in V$ such that for an edge $\langle p, q \rangle \in E$, $color(p) \neq color(q)$. We also want to use the *minimum number of colors possible* for G.

Possible lower bound: For the graph given above, it could be 2 or 3 (by inspection). For a general graph, it is 1 (for a graph with n singleton components).

Possible upper bound: For this graph, it is 4 (by the four color theorem). For a general graph, it can be as high as $n - 1$ (for a graph where all vertices are connected to each other).