To get full credit, you needed to first run a search algorithm (i.e. BFS) on G1 and G2 separately and then check the distance for the path through each possible edge, which the sum of the distances assigned to the endpoints of that edge from BFS + 1. You did not need to provide any pseudo-code or proof.

**Common Mistake #1: Adding all the edges into the graph and then running BFS.**
This algorithm is linear in the size of the inputs but it does not work, because it may provide the wrong path!

Example where this algorithm fails:
G1 is red, G2 is blue
instead of 4
Purple edges are in E'

![Diagram](image)

**Common Mistake #2: Adding each edge one at a time and then running BFS for each edge.**
This algorithm will produce the correct output, but it is not linear in the size of the inputs. Since BFS is $O(m+n)$ where $m$ and $n$ are the number of edges and nodes, and you are running it $|E'|$ times.
If $|E'| = p$, then this algorithm is $O( p^*(m+n) ) = O(pm + pn)$, which is not linear in the size of any of the inputs.

Another, less common, mistake was to assume that you get to choose the edges you want to create. You can’t just build a path from s to t because that edge might not be in the set of edges you are given.
Question 5 - DPV Problem 3.7(a)

To get full credit for this question, you needed to understand that the nodes of a bipartite graph can be divided into two sets such that no two nodes in the same set have an edge between them. This means a bipartite graph cannot have any odd-length cycles. This question also required that you argue WHY your algorithm would work.

You could check whether a graph is bipartite in a couple ways. The most common ways people used:
1) Try to color the graph using only 2 colors in a modified BFS, so no connected nodes have same color
2) Run BFS, then check whether any edges are between nodes with the same parity (even-even or odd-odd)
3) Run BFS, then check whether there is an edge between any two nodes with the same depth (distance from root)

The coloring method, or some variation of it, is most efficient because it halts the BFS as soon as it finds a node that cannot be colored without causing a conflict. The other two algorithms run a full BFS every time, and then have to run another subroutine over all the edges/nodes of the graph.

Common Mistake: Failing to argue WHY your algorithm is correct (-2 points)
Several people also described what their algorithm did but did not actually explain why it would work. You needed to explain in some way how your algorithm checked for the properties of a bipartite graph.
Question 6 - DPV Problem 3.11

To get full credit for this question, your algorithm needed to detect whether a graph contained a cycle including a specific edge AND return the cycle length if it did. To detect the cycle, you needed to show that there was more than one path from one node of that edge to the other.

The best solution is to remove edge, and then run BFS from one of the edge’s nodes to see if it assigned the other node a distance (not infinity). If BFS reaches the other node after the edge is removed, then there is another path between the nodes and there must be a cycle involving that edge. The length of the cycle is the distance assigned to the second node + 1 (to account for the remove edge).

Common Mistake #1: Not returning the length of the cycle (-2 points)

Common Mistake #2: Forgetting to account for the removed edge in the cycle length (-1 point)

Common Mistake #3: Not removing the edge first (or something similar)
If you run BFS but do not remove the edge first, it becomes very difficult to create and algorithm that correctly detects whether a cycle involving that edge exists.
Other Notes

PLEASE WRITE LEGIBLY.
I have horrible handwriting, so I understand it takes a little extra effort.
If I can't read one of your answers, it will receive a score of 0.

Refer to the syllabus for general grading information.

REMINDERS:
Lab 2 is due Wednesday, September 21 by the end of lab.
Homework 2 is due Friday, September 30th at the beginning of class.