Question 2

(a) Image included on next page

(b) The min-cut has value 7 and is made up of the partition s, a, e and f, b, c, d, t

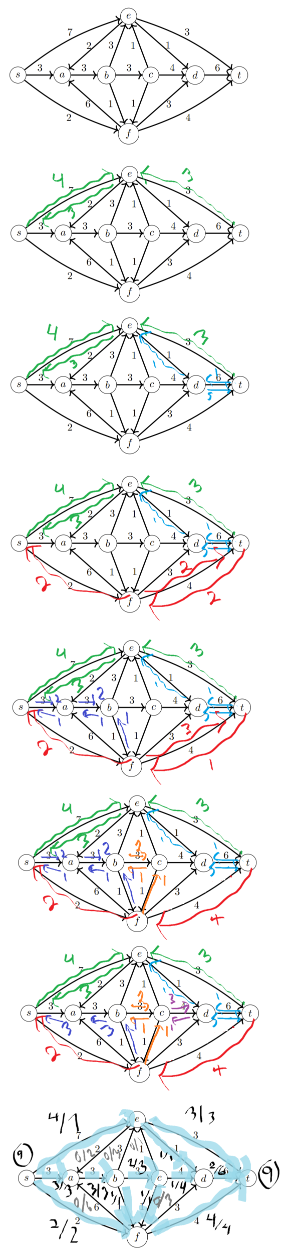

Figure 1: Residual graphs starting with before the while loop and ending with the final flow diagram.

Question 3

(a) One possible graph construction is to have a dummy source and a dummy sink node. From the source, we extend forward directed edges to dummy 'youth' and 'experienced' nodes that each have maximum capacity $(c * m) - m$. We then create nodes representing each senator (n total nodes) and extend forward directed edges from the 'youth' node to the senator node if the senator is youthful or from the 'experienced' node to the senator node if the senator is experienced. Each of these edges has maximum capacity of 5 since each senator can only serve on 5 committees maximum. We then create nodes representing each committee (m total nodes) and extend forward directed edges from the senator nodes to the committee nodes if the senator has deemed the committee acceptable. Each of these edges has a maximum capacity of 1. Finally, we connect each committee node to the dummy sink node with forward directed edges that have capacity c as each committee needs exactly c senators.

(b) The graph has 1+2+n+m+1 vertices, or n+m+4 vertices.

(c) Using our graph from part a), we can calculate a max-flow and check if it is equal to $c * m$. If the max-flow is less than $c * m$, then there is no valid assignment possible. If it is equal to $c * m$ (it cannot be greater because that is the maximum flow allowed by constraints in capacity), then we look at the capacities of the edges going from senator to committees. All edges have a maximum capacity of 1, and every edge that has full capacity (or 1/1) represents a senator being assigned to a specific committee.

(d) Since we limit each edge going from a committee to the sink node to capacity c, the maximum flow will force there to be c senators in each committee or less. If there are less, we return that a valid assignment is not possible, so it only returns an assignment when each committee from 1...m has c senators.

Since we constrained the edges from the source node to the 'youth' and 'experienced' nodes to capacity $(c * m) - m$, at least m of the senators must go into either the 'youth' or 'experienced' group. This means that as long as a valid assignment is possible, there are enough senators so that there can be 1 'youth' and 1 'experienced' senator in each committee.

Since we only made connecting edges between senators and the committees they find acceptable, there will never be a matching where a senator is in an unacceptable committee.

Since we made every maximum capacity from senator to committee 1 and the maximum capacity from the 'youth' and 'experienced' nodes to the senators 5, there can be a maximum of 5 1/1 full-capacity edges leaving a senator. Therefore, every senator will be at a maximum of 5 committees.

Question 4

(a) We can construct a new form reduction by using the provided pairs to generate a graph where there is an undirected edge between nodes i and j for the pair (i, j). We can then identify all the connected components in the graph and select an arbitrary root node in each connected component. Next, we can perform a modified 3-color algorithm on each of these connected component by passing in each newly chosen root node. Instead of standard colors, we use 0, 1, and 2 and if given a choice between 0, 1, or 2, we select the largest value. We can then iterate through every connected component and decide on which value belongs at each node, calculate the total number of 0s, and return this integer. This modified 3-color algorithm runs in polynomial time as well because we iterate through the graph once to select the largest value at each node and count the total number of 0s. Constructing the graph will take polynomial time and performing this new 3-coloring algorithm on each connected component will also take polynomial time. We know that the original 3-color algorithm can reduce to 3-SAT as we saw in class, meaning this modified version will as well since the only added component runs in polynomial time. This means the modified 3-color is at least NP-complete. Since the reduction from modified 3-color to this new algorithm runs in polynomial time as well, the problem is NP-hard.

(b) If our final algorithm returns a non-infinite number, then we can think of it as having returned YES to there being a possible assignment. If this is the case, then that means every individual run of the modified 3-color returned a non-infinite number (or YES) as well. If each individual connected component has a possible assignment, then the entire graph does. Therefore, if the algorithm returns YES, then the whole graph had a valid assignment.

If there is a possible assignment, then that means every individual connected component has a possible assignment. Then the modified 3-color algorithm run on each connected component returns an integer meaning an assignment is possible. Our final algorithm returns the sum of these individual numbers, so if there is a possible assignment, then our algorithm returns YES.

(c) No because we only proved this variation of the problem is NP-complete since we only solved the easy subsection of the problem where we only find its application onto trees. This does not show that P = NP since it is not applicable to non-special cases.

Question 5

(a) If we have a graph of 5 vertices in a row where each is connected to the next, then there are 5 vertices and 5 edges. In this example, there exists a Hamiltonian path where we start at vertex 0 and travel to each consecutive vertex until we reach vertex 4. We would return true because of this. However, since 1/3 of 5 is not an integer, there does not exist a Burr path, so taking a reduction from Hamiltonian would return True when the actual answer is False.

(b) A successful reduction can first check if the total number of vertices in the graph is a multiple of 3. If not, then we can return NO immediately. If it is, then we can create every possible subsection of the graph that contains exactly n/3 nodes (where n is the total number of nodes in the graph). We can then run the Hamiltonian Path solver on every one of these subsections and return true if any of these subsections' calls return true. If none do, we return false. This reduction can be achieved in polynomial time as we will have to perform it on every subsection once and can identify every subsections in polynomial time.

(c) If our final algorithm returns true, then that means there exists a run of the Hamiltonian Path solver on some subsection that returned true. If the Hamiltonian Path solver returned true, that means a path was successfully found, or that we were able to visit exactly 1/3 of the total vertices. Therefore, if the algorithm returns true, then a Burr path exists.

If there is a possible Burr path, then that means that from some node in the graph, we can travel to exactly 1/3 of the total vertices. Since each subsection we create has exactly 1/3 total vertices, we know that if the Hamiltonian Path solver returns true, then it was able to reach 1/3 of the total nodes. Therefore, our algorithm returns true if any one iteration of the solver returns true, or if there is a possible Burr path, our algorithm returns true.