H6.

- The assignment is due on Gradescope on Friday, March 5 at 5pm.
- You can either type your homework using LaTeX or scan your handwritten work. We will provide a LaTeX template for each homework. If you writing by hand, please fill in the solutions in this template, inserting additional sheets as necessary. This will facilitate the grading.
- You are permitted to discuss the problems with up to 2 other students in the class (per problem); however, *you must write up your own solutions, in your own words*. Do not submit anything you cannot explain. If you do collaborate with any of the other students on any problem, please do list all your collaborators in your submission for each problem.
- Similarly, please list any other source you have used for each problem, including other textbooks or websites.
- *Show your work.* Answers without justification will be given little credit.
- **NP-completeness guidelines:** You can cite and use the NP-completeness of any problem shown NP-complete in class or in K-T Chapter 8 (including the solved exercises, not that they are recommended for use). You are encouraged to consult Chap. 8 Sec. 10, "A Partial Taxonomy of Hard Problems", when looking for a good candidate problem to reduce to the problem being studied.

Problem 1 (30 points). Recall that an edge of a flow network is called a **bottleneck** edge if increasing its capacity results in an increase in the maximum flow. Given a flow network $G = (V, E)$ with integer capacities $c_e > 0$, and a maximum flow f between s and t in G give an algorithm running in time $O(|V| + |E|)$ that finds all bottleneck edges in G. Prove that your algorithm is correct and that it runs in the required running time.

Solution: Your solution goes here.

Lewis Arnsten, Collaborators: David Wu, Brian McManus

Professor Drucker granted me an extension for this assignment.

Problem 1:

Given a flow network G = (V, E) with integer capacities $C_e > 0$, and a maximum flow *f* between s and t in G, we must find all bottleneck edges in G. To do so, we will use Ford-Fulkerson to find our final disconnected residual graph. This graph contains two sets of nodes: the set reachable from the source and the set reachable from the sink. The edges that connect these two sets are bottleneck edges.

Correctness: The edges produced by this algorithm must necessarily be bottleneck edges, otherwise the residual graph used would not be the final residual graph. In other words, there would be remaining capacity to run another iteration of Ford-Fulkerson. Thus, increasing the capacity of one of the found edges would not increase the maximum flow.

Time complexity: When given a value of flow, Ford-Fulkerson can construct a residual graph in O(V+E). Consider the following algorithm for finding our final residual graph. Given a flow network G = (V, E) with integer capacities $C_e > 0$, and a flow f:

```
Create an empty graph residual = (V, empty)
For each (i, j) in E:
    Create a directed "forward" edge (i, j) in residual with weight $C_e - f_e$
    Create a directed "backward" edge (j, i) in residual with weight $f_e$
Return residual
```

Since we need to build our empty residual graph and then iterate over all edges, this algorithm is O(V + E).

Problem 2 (35 points). You are running a tutoring service and need to match n tutors with n students over m days. Each tutor would like to come in once to tutor a student and each student would like to come in once to be tutored. However, there are several constraints you must obey:

1. Each tutor t_i can only come in on a subset T_i of the days.
2. Each student s_j can only come in on a subset S_j of the days.
3. Due to space limitations, on day k you can have at most c_k meetings between students and tutors.

Give a polynomial-time algorithm to find a way to schedule meetings between the tutors and students (or determine that it is impossible to do so); prove that the algorithm is correct and runs in polynomial time. When proving efficiency, please state an explicit running time using $O(\cdot)$ notation.

Note: 80% partial credit will be given if you solve the problem when there is no limit to how many people can meet on a given day.

Solution: Your solution goes here.

Problem 2:

Let t = {t_1, t_2, ..., t_n} be the set of tutors, and s = {s_1, s_2, .., s_n} be the set of students. Each tutor can only come in on a subset T_i of days and each student can only come in on a subset S_j of days. We will create a graph G to find a matching between our tutors and students as follows:

For all n tutors in t we will create a node in G which we connect to the source with an edge of capacity 1. Then, for all m days we create nodes, and we create edges of capacity 1 between each tutor and all the days when they can come in (all days in T_i). We then create another series of nodes representing the days (containing m nodes again) which we will connect to our previous layer of days directly (a day k in our first layer of days will be connected to the same day k' in our second layer). Each edge connecting these two layers will have a capacity C_k. Then we will create nodes for each student in s and we will create edges of capacity 1 between each student and all the days (in the previous layer) S_j when they can come in. Finally, we will create edges of capacity 1 connecting each of our students to our sink. From this graph G, we can find the maximum flow via Edmonds-Karp algorithm, producing a way to schedule meetings between the tutors and students.

Correctness: Based on how we constructed G, it is necessarily true that the maximum s-t flow in G will yield the maximum number of meetings that can be scheduled between tutors and students. Theorem: the max cardinality of a matching in G is equal to the value of the maximum flow in G' where G is a bipartite graph defined as G = (L ∪ R, E) and G' = (L ∪ R ∪ {s,t}, E'). Our graph is defined as G = (t ∪ T ∪ S ∪ s ∪ {s,t}, E). Our graph is similar to a bipartite graph, however we have added two middle layers to account for the space limitations on a day k. Since our flow is still limited by the unit capacity edges connecting our source to our tutor nodes (as well as those connecting tutors to the first layer of days T), we are sending 1 unit of flow along each of n paths. Although these flows may combine on the edges connecting our middle layers of days (T ∪ S, E), they are forced to separate into flows of value 0-1 when connecting the second layer of days to the students (S ∪ s, E). As such, it is true that the ideal scheduling between tutors and students is equal to the value of the maximum flow in G (this is also seen to be true since the minimum cut of our graph G is equal to that of G' = (t ∪ s ∪ {s,t}, E'). Further, by adding the two middle layers of days and assigning edges between them a capacity C_k, we ensure that a day k can have at most C_k meetings between students and tutors. Therefore, by finding the maximum flow by Edmonds-Karp we are finding the best way to schedule meetings between students and tutors.

Time complexity: After we have constructed our graph, we find the maximum flow using Edmonds-Karp algorithm. Edmonds-Karp is an implementation of Ford-Fulkerson which uses breadth-first search to find augmenting paths. The complexity of BFS is O(E), and the complexity of Ford-Fulkerson is O(E*F), as the maximum amount of flow that we might have to route is F and routing one augmenting path requires time proportional to the size of the graph E. An upper bound for our implementation of Edmonds-Karp is therefore $O(V*E^2)$--meaning our algorithm runs in polynomial time.

PROBLEM 3 (35 points). Consider the decision problem 1-INTERSECT. Given subsets $S_1, ..., S_m$ of $[n]$, return YES if there is a subset $A \subseteq [n]$ such that for all $j \in [m]$, we have $|S_j \cap A| = 1$. Otherwise, return NO. As an example, let $n = 6$ and consider the instance consisting of the following subsets.

$$S_1 = \{1,2\}, \quad S_2 = \{1,5,6\}, \quad S_3 = \{3,4,5,6\}, \quad S_4 = \{2,3,6\}, \quad S_5 = \{2,4,6\}$$

The correct output for this instance is YES since choosing $A = \{2,5\}$ will intersect each S_j once.

Prove that 1-INTERSECT is NP-complete.

Hint: Recall the 3-COLORING problem.

Solution: Your solution goes here.

Problem 3:

1-Intersect set = a subset A on [n] that shares exactly one intersection with each subset $S_1, ..., S_m$ of $[n]$.

We claim 1-Intersect set is NP-complete. The proof follows.

1. 1-Intersect set in NP: Given subsets $S_1, ..., S_m$ of $[n]$, the 1-intersect set is constructed by choosing a value which appears in each S_j exactly once. It is clear that this behavior can be implemented in polynomial time.

2. 1-Intersect set is NP-hard: We will show that some NP-complete L is polytime-reducible to 1-Intersect set. In this case we choose L = 3-D Matching.

In 3-D matching, given disjoint sets X, Y, and Z, each of size n, and given a set $T \subseteq X \times Y \times Z$ of ordered triples, we find a set of n triples in T so that each element of $X \cup Y \cup Z$ is contained in exactly one of these triples.

We can define a reduction which takes as input the disjoint sets X, Y, and Z, each of size n, and a set $T \subseteq X \times Y \times Z$ of ordered triples. To create an instance of one-intersect, we create the set, S_1, of all triples containing the first element of x, the set S_2, of all triples containing the second element of x, and so on for all elements of X, Y, and Z. This will produce sets $S_1, ..., S_m$ where m = 3*n.

It is clear that this reduction can be performed in polynomial time. Now we show the reduction's correctness property, in two parts.

a) Claim: The set of triples produced from 3-D matching given our reduction is exactly the subset A that satisfies 1-intersect. This is true because each element in X, Y, and Z will appear in exactly one triple of A. Thus, the intersection of A and any subset $S_1, ..., S_m$ of $[n]$ will be equal to one.

b) Claim: Conversely, if we get a yes answer from one intersect. Then, based on how we constructed an instance of one-intersect (we defined S_i as the set of all triples containing the ith element of X, Y, or Z), 3-D Matching produces a set of triples such that each element of $X \cup Y \cup Z$ is contained in exactly one of these triples. This is true because, when S_i intersected with A equals 1, it must be true that each element of X, Y, and Z is only contained in one triple.

It follows from the above that 3-D Matching polytime reduces to 1-intersect, which is therefore NP-hard. We have completed the proof that 1-intersect is NP-complete.