CS 6363.003 Spring 2021 Official Solutions
**Homework 5 Problem 1**

A given flow network $G = (V, E)$ with source $s$, sink $t$, and capacity function $c : E \to \mathbb{R}_{\geq 0}$ may have more than one minimum $(s, t)$-cut.

(a) Let $(S, T)$ and $(S', T')$ be two minimum $(s, t)$-cuts in $G$. Prove that $(S \cap S', T \cup T')$ and $(S \cup S', T \cap T')$ are also minimum $(s, t)$-cuts in $G$.

**Solution:** Let $f^*$ be an arbitrary maximum $(s, t)$-flow in $G$. Consider any edge $u \to v$ with $u \in (S \cap S')$ and $v \in (T \cup T')$. Suppose without loss of generality that $v \in T$. Vertex $u \in S$, and $(S, T)$ is a minimum $(s, t)$-cut, so $f^*$ saturates $u \to v$. Similarly, consider any edge $w \to x$ with $w \in (T \cup T')$ and $x \in (S \cap S')$. Suppose without loss of generality that $w \in T$. Vertex $x \in S$, and $(S, T)$ is a minimum $(s, t)$-cut, so $f^*$ avoids $w \to x$. We see $f^*$ saturates every edge $u \to v$ with $u \in (S \cap S')$ and $v \in (T \cup T')$ and avoids every edge $w \to x$ with $w \in (T \cup T')$ and $x \in (S \cap S')$, so $(S \cap S', T \cup T')$ is a minimum $(s, t)$-cut.

A nearly identical argument implies $(S \cup S', T \cap T')$ is a minimum $(s, t)$-cut as well. ■

**Rubric:** 4 points total.

(b) Describe and analyze an efficient algorithm to determine whether $G$ contains a unique minimum $(s, t)$-cut.

**Solution:** We begin by computing a maximum $(s, t)$-flow $f^*$ in $G$ using Orlin's algorithm. Let $G_{f^*}$ be the residual graph of $G$ with respect to $f^*$. We compute the set $S$ of vertices reachable from $s$ in $G_{f^*}$ using a breadth-first search from $s$. Then, we compute the set $T$ of vertices *that can reach* $t$ in $G_{f^*}$ using a breadth-first search from $t$ *in the modification of* $G_{f^*}$ *that reversals all of its edges*. If $S \cup T = V$, we report $G$ contains the unique minimum $(s, t)$-cut $(S, T)$. Otherwise, we report $G$ contains multiple minimum $(s, t)$-cuts.

Let $(S', T')$ and $(S'', T'')$ be two minimum $(s, t)$-cuts in $G$. As shown in part (a), $(S' \cap S'', T' \cup T'')$ is a minimum $(s, t)$-cut as well, and all edges going between $S' \cap S''$ and $T' \cup T''$ are saturated or avoided by $f^*$. As explained in class, $G_{f^*}$ contains no edges going from $S' \cap S''$ to $T' \cup T''$. By iteratively applying this argument to all minimum $(s, t)$-cuts in $G$, we conclude the set $S$ defined above contains exactly the vertices in the intersection all of "$s$ sides" of all minimum $(s, t)$-cuts of $G$. Similarly, set $T$ is the intersection of all "$t$ sides". If there are multiple minimum $(s, t)$-cuts, then there exists a vertex $v$ that appear in an $s$ side at least once, implying $v \notin T$, and appears in a $t$ side at least once, implying $v \notin S$. On the other hand, if the minimum $(s, t)$-cut is unique, then we know from class that it is $(S, T)$.

It takes $O(VE)$ time to runs Orlin's algorithm and $O(E)$ time to do both searches. The algorithm takes $O(VE)$ time total. ■

**Rubric:** 6 points total: 3 points for the algorithm; 1.5 points for justification, 1.5 points for running time analysis.

CS 6363.003 Spring 2021 — Official Solutions
**Homework 5 Problem 2**

Describe and analyze an algorithm to determine whether a given student can graduate. The input to your algorithm is the list of $m$ requirements (each specifying a subset of the $n$ courses and the number of courses that must be taken from that subset) and the list of courses the student has taken.

**Solution:** We reduce to maximum $(s,t)$-flow as follows. We begin by building a flow network consisting of a directed graph $G = (V, E)$ and a capacity function $c : E \to \mathbb{R}_{\geq 0}$. Graph $G$ has as vertices

- a source vertex $s$,
- a set of $n$ *course vertices* $p_1, \dots, p_n$, one for each course $i$,
- a set of $m$ *requirement vertices* $r_1, \dots, r_m$, one for each requirement $j$, and
- a sink vertex $t$.

It also has as edges

- an edge $s{\to}p_i$ of capacity $c(s{\to}p_i) := 1$ for each course $p_i$ the student has taken,
- an edge $p_i{\to}r_j$ of capacity $c(p_i{\to}r_j) := 1$ for each course $i$ that partially satisfies a requirement $j$, and
- an edge $r_j{\to}t$ for each requirement $j$ where the capacity $c(r_j{\to}t)$ of the edge is set to the number of courses needed to satisfy the requirement.

We compute a maximum $(s,t)$-flow $f^*$ in $G$ and report the student can graduate if and only if every edge $r_j{\to}t$ is saturated by $f^*$.

Indeed, suppose the student can graduate, and consider an assignment of courses to graduation requirements that exactly meets the number of courses needed for each requirement. Let $f$ be a flow where $f(s{\to}p_i) = 1$ for each course $i$ the student uses to satisfy any graduation requirement, $f(p_i{\to}r_j) = 1$ for each course $i$ used to satisfy requirement $j$ in particular, $f(r_j{\to}t)$ equals the number of courses needed to satisfy requirement $j$, and $f(e) = 0$ for all other edges of $G$. Each course vertex $p_i$ has at most one incoming and one outgoing unit of flow, because course $i$ can be used to satisfy at most one requirement. There is no incoming or outgoing flow for $p_i$ if the student either did not take course $i$ or the course wasn't needed to satisfy a requirement. Each requirement vertex $r_j$ has exactly $x_j$ incoming edges with one unit of flow and a single outgoing edge with $x_j$ units of flow where $x_j$ is the number of courses needed to satisfy requirement $j$. Therefore, flow $f$ satisfies conservation constraints. For each edge $e$, we have $f(e) = 0$ or $f(e) = c(e)$, so capacity constraints are satisfied as well.

Conversely, suppose the maximum flow $f^*$ saturates every edge $r_j{\to}t$. We will describe a way to satisfy the student's graduation requirements. We may assume $f^*$ is integral. Each requirement vertex $r_j$ has $x_j$ outgoing units of flow where $x_j$ is the number of courses needed to satisfy requirement $j$. By conservation constraints, it must also have $x_j$ incoming units of flow distributed among $x_j$ edges $p_i{\to}r_j$ each carrying exactly one unit. We'll assign each of the $x_j$ corresponding classes $i$ to partially satisfy requirement $j$. Each vertex $p_i$ has at most one unit of incoming flow, and therefore at most one unit of outgoing flow. We may conclude each class $i$ is being used to satisfy at most one requirement.

Graph $G$ has $O(n+m)$ vertices and at most $O(nm)$ edges. The $(s,t)$-cut $(\{s\}, V \setminus \{s\})$ has capacity at most $n$, so $|f^*| \leq n$. We can simply run the Ford-Fulkerson augmenting path approach with arbitrary choices of augmenting path for a total running time of $O(E|f^*|) = O(n^2 m)$. ■

**Rubric:** 10 points total: 5 points for the algorithm; 3 points for justification, 2 points for running time analysis.

CS 6363.003 Spring 2021 — Official Solutions

**Homework 5 Problem 3**

DNF-SAT asks, given a boolean formula in disjunctive normal form, whether that formula is satisfiable.

(a) Describe a polynomial-time algorithm to solve DNF-SAT.

**Solution:** We need only satisfy a single term for the whole formula to be satisfied. Each term is a conjunction of literals, so it is possible to satisfy a term if and only if no two literals in the term are negations of one-another. Therefore, for each term, we sort the literals by variable symbol and then do a linear scan to see if any pair of consecutive literals contradict. If no contradictions are found in even a single term, we return TRUE. Otherwise, we return FALSE.

The sorting of disjoint terms takes $O(n \log n)$ time total for a formula with $n$ literals total. ■

**Rubric:** 6 points total: 3.5 points for the algorithm; 1.5 points for justification, 1 point for running time analysis. Any polynomial time algorithm is worth full credit.

(b) What is the error in the argument that $\mathrm{P} = \mathrm{NP}$?

**Solution:** This reduction may take exponential time! Indeed, suppose we are given a 3-CNF formula with $k$ clauses. Naively applying the distributive law would require the creation of $3^k$ terms.

We need a polynomial time reduction to conclude DNF-SAT is NP-hard, so we have not shown $\mathrm{P} = \mathrm{NP}$. ■

**Rubric:** 4 points total.

CS 6363.003 Spring 2021 Official Solutions
**Homework 5 Problem 4**

Let's practice some NP-hardness proofs!

(a) Prove that PebbleDestruction is NP-hard.

**Solution:** We will perform a reduction from the known NP-hard problem Hamiltonian path in an undirected graph. (A Hamiltonian path is a path that contains every vertex of the graph.)

Suppose we are given an undirected graph $G = (V, E)$, and we wish to know if $G$ contains a Hamiltonian path. We construct a graph $G' = (V', E')$ by augmenting $G$ as follows: First, we add a single vertex $s$ to $V$, so $V' = V \cup \{s\}$. We then add edges from $s$ to all vertices of $G$, so $E' = E \cup \{sv : v \in V\}$. Define pebble counts $p : V' \to \mathbb{Z}_{\geq 0}$ so that $p(s) = 2$ and $p(v) = 1$ for all other vertices $v \in V$ of the original graph $G$. Given an algorithm for PebbleDestruction, we could run it on graph $G'$ and pebble counts $p$ and report there is a Hamiltonian cycle if and only if there is a sequence of pebbling moves that remove all but one pebble from $G'$. This reduction takes $O(V)$ time outside the PebbleDestruction algorithm, as we only need to add $O(V)$ edges and describe $p$.

Suppose there is a Hamiltonian path $P$ in $G$, and let $P'$ be $P$ prepended with $s$. We can write $P'$ as a sequence of vertices $\langle s = v_0, v_1, \ldots, v_{|V|}\rangle$ where consecutive pairs $v_{i-1}, v_i$ are adjacent in $G'$. Consider the following sequence of pebbling moves. For each $i$ from 0 to $|V|-1$, we may inductively assume that vertices $\{v_0, \ldots, v_{i-1}\}$ each have 0 pebbles, vertex $v_i$ has two pebbles, and vertices $\{v_{i+1}, \ldots, v_{|V|}\}$ each have one pebble. We remove the two pebbles from $v_i$ and place one additional pebble on $v_{i+1}$. Finally, we do one last pebbling move where we remove the two pebbles from $v_{|V|}$ and place one pebble on $s$. After this move, $s$ has a single pebble and all other vertices of $G'$ have no pebbles.

Now, suppose there is a sequence of pebbling moves that remove all but one pebble from $G'$. We may assume inductively that just before any move, there is exactly one vertex with two pebbles and all others have either zero or one pebbles. The vertex with two pebbles ends the move with zero pebbles. If the move places a pebble on some vertex $v$ with exactly one pebble, then $v$ becomes the only vertex with two pebbles. However, if the move places a pebble on a vertex $v$ with zero pebbles, then there no longer exist any vertices with two pebbles and that must have be the final move. Finally, if any vertex $v$ of $G$ never receives a pebble, then it ends the sequence with its original pebble and at least one more pebble remains either on $s$ or on the last vertex to receive a pebble.

We conclude that each move removes two pebbles from a distinct vertex, every vertex becomes the two pebble vertex for exactly one move, and consecutive pairs of these two pebble vertices are adjacent in $G'$. Therefore, the vertices that have two pebbles, in order, form a Hamiltonian path $P'$ in $G'$ that begins with $s$. By removing $s$ from $P'$, we get a Hamiltonian path $P$ in $G$. ■

**Rubric:** This reduction was more subtle than Kyle anticipated and he misled several students during office hours into thinking there was a very simple reduction from Hamiltonian cycle. Therefore, he is going to give 5 extra credit points to everybody, including those who did not submit a solution.

(b) Prove that StoneSolitaire is NP-***complete***.

**Solution:** First, StoneSolitaire is in NP, because we can certify a positive solution by just showing where the surviving stones go. It is straightforward to confirm whether the stones are consistent with the input and satisfy the two conditions in polynomial time.

To show NP-hardness, we will perform a reduction from the known NP-hard problem 3Sat. Suppose we are given a 3CNF formula $\Phi$, and we wish to know if there is an assignment to its variables that makes it evaluate to True.

We define an instance of StoneSolitaire as follows. We have one column for each of the $m$ variables of $\Phi$. For each clause $C$ of $\Phi$, we do the following. If $C$ contains both a variable and its negation, we continue to the next clause. Otherwise, we add a row to our StoneSolitaire instance. For each positive literal $a$ in $C$, we add a blue stone at the intersection of $a$'s column and $C$'s row. For each negative literal $\bar{a}$ in $C$, we add a red stone at the intersection of $a$'s column and $C$'s row. After we finish looping through all the clauses, we report $\Phi$ is satisfiable if and only if the StoneSolitaire instance can be solved. If $\Phi$ contains $k$ clauses, then the reduction takes $O(k)$ time outside of solving the StoneSolitaire instance.

Now, suppose $\Phi$ has a satisfying assignment. We can solve the StoneSolitaire instance as follows. For each variable $a$, we remove all red stones from $a$'s column if $a$ is assigned True. Otherwise, we remove all blue stones from $a$'s column. Clearly, no column still has stones of both colors. Further, each row still contains the one or more stones corresponding to the True literal for the row's clause.

Suppose the StoneSolitaire instance can be solved. We can find a satisfying assignment to the variables of $\Phi$ as follows. For each column of the solved StoneSolitaire instance, we do the following. If the column contains only blue stones, we set the column's variable to True. Otherwise, we set the column's variable to False. For each clause $C$ in $\Phi$, either $C$ contains both a literal and its negation, in which case $C$ is trivially satisfied, or there is a row for $C$ that contains at least one stone. If the stone is blue, then we set the variable for the stone's column to True, and the clause is satisfied. If the stone is red, we set the variable for the stone's column to False, and the clause is still satisfied.

Because the problem is both in NP and NP-hard, we conclude StoneSolitaire is NP-complete. ■

**Rubric:** 5 points total: 1 point for arguing inclusion in NP; 1.5 points for the reduction; 0.5 points for running time; 1 point for each direction of the if and only if proof.