CSCI 570 Spring 2020 Xinrui Ying (USCID 1092350440)
Homework 5 Problem

Solution:

1. Initially, they would have $1 \times 500 + 2 \times 300 + 2 \times 200 = 1500$ feet of framing wood and $3 \times 500 + 2 \times 300 + 1 \times 200 = 2300$ feet of cabinet wood. Therefore, after the shortage, they will have 1600 feet of framing wood and 1700 feet of cabinet wood.

   Let x_1, x_2, x_3 be the number of three types of couches they made that month, the objective function is:

   $$max(10x_1 + 8x_2 + 5x_3)$$

   subject to

   - $x_1 + 2x_2 + 2x_3 \leq 1600$
   - $3x_1 + 2x_2 + x_3 \leq 1700$
   - $x_1 \geq 0$
   - $x_2 \geq 0$
   - $x_3 \geq 0$

2. The dual of the problem would be

$$min(4y_1 + 6y_2 + 3y_3 - 3y_4 + 8y_5)$$

subject to

- $y_1 + 2y_2 - y_3 + y_4 + y_5 \geq 3$
- $-y_1 + y_2 + y_5 \geq 2$
- $y_1 + 3y_2 + 2y_3 - 2y_4 + y_5 \geq 1$
- $y_i \geq 0 \ \forall i \in [1,5]$

3. Let $x_{i,j} = 1$ if s_i selected f_j and $x_{i,j} = 0$ otherwise for all $i \in [1, n]$ and $j \in [1, m]$. Our objective function is:

$$min(\sum_{i=0}^{i=n} \sum_{j=0}^{j=m} x_{i,j} \cdot f_j)$$

subject to

- $\sum_{j=0}^{j=m} x_{i,j} = 1 \ \forall i \in [1, n]$
- $\sum_{j=0}^{j=m} x_{i,j} \cdot x_{i',j} = 0 \ \forall (s_i, s_{i'}) \in E$
- $x_{i,j} \in \{0, 1\}$

4. (1) The statement is $False$. $A \leq_p B$ means A is at most as hard as B. If B is in $NP-hard$, then A can also be in P, and P is not in $NP-hard$.

   (2) The statement is $True$. $A \leq_p B$ means A is at most as hard as B. If B is in NP, then A must also be in NP.

   (3) The statement is $True$. Because $2-SAT$ has a polynomial algorithm and $3-SAT$ is in $NP-complete$, if a $NP-complete$ can be reduced to a P problem, then $P = NP$.

   (4) The statement is $True$. Every NP problem has a exponential-time algorithm because by definition, it can be decided by a non-deterministic Turing machine in polynomial time.

   (5) The statement is $False$. Only easy problem can be reduced to harder problem, doesn't work otherwise. Otherwise, all problems can be reduced from each other.

5. We loop through variables x_i in 3—SAT, for each x_i, we set x_i to $True$, and the x_i is a literal in a clause C we remove the clause C from the 3—SAT, if $\neg x_i$ is in a clause C, we simply remove $\neg x_i$ in that clause. Then we run the algorithm to determine if it has a satisfying assignment. If it has, we know that x_i is True, we continue; if it doesn't have, we also know x_i is false, we add back the removed literal in last step, and the $\neg x_i$ is a literal in a clause C we remove the clause C from the 3—SAT, if x_i is in a clause C, we simply remove x_i in that clause.

   Let $p(n)$ be the time for checking if it has a satisfying assignment. The algorithm will take $O(n * (p(n) + n))$. Each iteration we will spend $p(n)$ check if it's valid and $O(n)$ time for removing or adding back literals and clauses.

6. First we show that the problem is in NP, if we are given the arrangement of how to put the products into boxes, we can quickly check if k box can hold all products. This will take linear time to check.

   Then, we will show that it can be reduced from Set Partition Problem. Given $x_1, x_2, ..., x_n$, we assign x_i to p_i accordingly. Let $W = \frac{1}{2}\sum_i x_i$ and $k = 2$.
   If there exists a partition that $\sum_{i \in S} x_i = \sum_{i' \notin S} x_{i'}$, then means we can fit $p_i \ \forall i \in s$ in a box since $\sum_{i \in S} x_i = \frac{1}{2}\sum_i x_i \leq W$, similarly, we can put all $p_i \ \forall i \notin S$ in the second box because $\sum_{i' \notin S} x_{i'} \leq W$. Therefore the problem is also true.
   If there exists a way to put products in two boxes, then each box must be full since $2W = \sum_i x_i$, then for all i in box 1 and i' in box 2, $\sum_i p_i = \sum_{i'} p_{i'} = W$. Then, let the i in the first box be set S, we have $\sum_{i \in S} x_i = \sum_{i' \notin S} x_{i'}$. Thus, the Partition Problem is also true.

   We showed that the problem is both in NP and NP-Complete problem can reduce to it, so is a NP-Complete problem.

7. First, we want to show that longest-path problem is in NP. Given a solution, we can simply traverse the path and keep track the vertices to check if it's true. This takes linear time.

   Then, we show that we can reduce from Hamiltonian path to longest-path problem. Let $G = (V, E)$ and $V = k + 1$, there exists a Hamiltonian path if and only if there exist a longest-path of length k.
   If there exists a Hamiltonian path, then the path length will be $V - 1 = k$, since Hamiltonian path only visits each vertex once and it visits all the vertices. Therefore it visits in total of $k + 1$ vertices and the path will be length k. Therefore, we find a longest-path of length k.
   If there exist a longest path of length k, we must know there will be in total of $k + 1$ different vertices visited and no vertices are visited twice. Since $V = k + 1$, then this path is also a Hamiltonian path.

   We showed that longest-path is both in NP and NP-Complete problem can reduce to it. Longest-path problem is a NP-Complete problem.

8. The algorithm 1 is as following:

   - First we select any city c_i and put it in C.
   - Loop this step for $k-1$ times, we select the s that maximize $dist(s, C)$ and we put it in C.

   Now, we show that the r of C is at most $2r^*$ where r^* is the optimal distance by contradiction.

   We will first introduce an algorithm 2 which given value of r', find k centers which have less than $2r'$. The algorithm is simply, initialize a set S contains all cities and a set $C = \emptyset$. First we select a random city s, we add it to C and remove it from S. We than will need to move all cities t in S that has $dist(t, s) > 2r'$. Loop this iteration for k times and if S is empty then we will return C as result else it's impossible to have such a set C.

   By contradiction, we assume that $r > 2r^*$, then we know there will be a city c' that has $dist(c', C) > 2r^*$, because from algorithm 1 for finding C, at each iteration, we are selecting city t that maximize $dist(t, C)$, then $dist(t, C) > dist(c', C) > 2r^*$ for all iterations. Also, our algorithm 1 acts the same for using algorithm 2 to find k centers that has maximum distance $2r^*$. Then, based on the algorithm 2, the city c' will still be in set S and therefore no set of size k will have distance within $2r^*$. Hence, we won't be able to find a set of size k has distance within r^*. This contradicts with r^* being the optimal distance. We have a conflict, then $r \leq 2r^*$ and our algorithm is 2—approximation.

   ■