P1) Given a connected graph $G = (V, E)$ with $n$ vertices and $m$ edges where every edge has a positive weight $w_e > 0$, for any pair of vertices $u, v$ define $d(u, v)$ to denote the length of the shortest path from $u$ to $v$ in $G$.

(a) Prove that $d(., .)$ is a metric, namely it satisfies the following three properties: (i) $d(u, v) \geq 0$ for all $u, v$ and $d(u, v) = 0$ only when $u = v$. (ii) $d(u, v) = d(v, u)$ for all vertices $u, v \in V$. (iii) $d(u, v) + d(v, w) \geq d(u, w)$ for all $u, v, w \in V$.

(b) Let $d^* := \max_{u,v \in V} d(u, v)$ denote the longest shortest path in $G$. Design an $O(m \log(n))$ time algorithm that gives a 2-approximation to $d^*$, i.e., your algorithm should output a number $\tilde{d}^*$ such that

$$\tilde{d}^* \leq d^* \leq 2\tilde{d}^*.$$ 

In this part you can use the Dijkstra’s algorithm which finds the shortest path from a given vertex $s$ to all vertices of $G$. We will discuss Dijkstra’s algorithm later in the course. You can further use this algorithm runs in $O(m \log n)$.

Part a) (i) $d(u, v) \geq 0$ holds since all edges have non-negative weights. (ii) $d(u, v) = d(v, u)$ since the graph is undirected, any path from $u$ to $v$ is also a path from $v$ to $u$. (iii) holds by composing paths: Any path from $u$ to $v$ can be concatenated with a path from $v$ to $w$ (possibly deleting repeated vertices) to obtain a path from $u$ to $w$. This gives a candidate path from $u$ to $w$ and $d(u, w)$ is the shortest one that may or may not pass $v$ along the way.

Part b) We run the Dijkstra’s algorithm from an arbitrary vertex $v$. Let $u$ be the farthest vertex from $v$ in the output of Dijkstra’s algorithm. we output $d(u, v)$. Let $u^*, v^*$ be the farthest vertex in $G$, and $d^* = d(u^*, v^*)$. We need to show that

$$d(u, v) \leq d^* \leq 2d(u, v).$$

The first inequality, $d(u, v) \leq d^*$ follows by optimality of $d^*$, i.e., that $d^*$ is the largest shortest path among all pairs including $u, v$.

To prove the second inequality we used the triangle inequality of $d(., .)$; namely:

$$d(u^*, v^*) \leq d(u^*, v) + d(v, v^*) \leq d(u, v) + d(v, u) = 2d(u, v).$$

P2) Suppose you are given $n$ coins with value $v_1, \ldots, v_n$ dollars, and you want to change $S$ dollars. You can assume $v_i \neq v_j$ for all $i \neq j$. Design a polynomial time algorithm that outputs the
number of ways to change $S$ dollars with the given $n$ coins. For example, if for values 1, 2, 3, 4 we can change 6 in 2 ways as follows:

$$2 + 4, 1 + 2 + 3$$

**Solution:** I start by writing a wrong DP: Let $OPT(S)$ be the number of ways to change $S$ dollars with coins $v_1, \ldots, v_n$. One can say either $OPT$ uses $v_1$ or $v_2, \ldots$ or $v_n$ so one can write

$$OPT(S) = \sum_i OPT(S - v_i).$$

This is wrong, why? Because it double counts. For example, say $v_1 = 1, v_2 = 2$ and $S = 3$. Then, we write $OPT(3) = OPT(1) + OPT(2)$, and since $OPT(1) = 1, OPT(2) = 1$, we get $OPT(3) = 2$. But the write answer is $OPT(3) = 1$. So, where is the mistake? We are double counting 1, 2 and 2, 1.

The right way to do it is to do a two-dimensional $OPT$: Let $OPT(s, i)$ be the number of ways to change $s$ dollars using only coins $v_1, \ldots, v_i$. **Base Case:** $OPT(0, i) = 1$ and $OPT(s, 0) = 0$ for any $s > 0$.

Now, we do the inductive step: We ”guess” whether coin $v_i$ used in $OPT(s, i)$. Note that $v_i$ can be used only if $v_i \leq s$. If we use coin $v_i$ then we need to change the rest of $s - v_i$ dollars using coins $v_1, \ldots, v_{i-1}$. So, we claim that $OPT(s, i)$ is the sum of all of these possibilities.

$$OPT(s, i) = \begin{cases} 
OPT(s - v_i, i - 1) + OPT(s, i - 1) & \text{if } v_i \leq s \\
OPT(s, v - i) & \text{o.w.}
\end{cases}$$

First, we are not double counting in the above calculation: This is because whether we put $v_i$ in or out we are counting two different approaches that to change $s$ dollars so we don’t double count. Second, we count all possibilities because $OPT$ other uses or doesn’t uses $v_i$.

The algorithm follows:

```
for s = 0 → S do
    Set $M[s, i] = empty$ for all $0 \leq i \leq n$
end

Function $OPT(s, i)$
    If $s = 0$ return 1 else if $i = 0$ return 0
    If $M[s, i] \neq empty$ return $M[s, i]$
    If $v_i \leq s$, $M[s, i] = OPT(s - v_i, i - 1) + OPT(s, i - 1)$ else $M[s, i] = OPT(s, i - 1)$.
    return $M[s, i]$

OPT(S,n).
```

Section 5-2