Theory of Computation, CSCI 438 spring 2022
Polynomial Time, pg. 284-291, April 25

1. PATH = {<G, s, t> | G is a directed graph that has a directed path from s to t}
   Prove that PATH$\in$P

   Prove that PATH$\in$P
   Naïve way – exam all potential paths in G. Say that G has n nodes, this would take n^n time.

   Students may say "use a breath first search". Breath first search for the worst case, a complete graph, starts at s. (Let's use n as the number of edges.) At the next level it has n-1 branches (assuming s shared an edge with every other vertex). Now we need to work with each vertex, which could share a branch with every other vertex. This can give exponential time.

   Better: Start by marking node s. Repeatedly mark all nodes which are reachable from a marked node. This has a polynomial time running time.

   M where $\mathcal{L}$(M)=PATH
   M = "On input <G,s,t> where G is a directed graph with nodes s and t:
   1. Mark s as visited.
   2. Repeat until the number of nodes is not increasing:
      Go through the list of edges (x, y) in G, marking any node y when x was already marked.
   3. If t is marked accept; otherwise reject."

For analysis can have n be the number of nodes or the number of edges. I'll let n be the number of edges. Steps 1 & 3 are 1 step each. Step 2 requires n steps for one iteration. At most it will executed n times (really considerably less). Thus this runs in $O(n^2)$ time.

2. Two integers are relatively prime (or coprime) if there is no integer greater than one that dives them both.

   RELPRIME = {<x, y> | x and y are relatively prime}
   Prove that RELPRIME$\in$P

   Prove that RELPRIME$\in$P
   Naïve way – search through all possible divisors of both numbers and accept if none are greater than 1. However, the magnitude of a number represented in binary, or in any other base k notation for k≥2, is exponential in the length of its representation. Therefor this algorithm has exponential running time.

   Better: Use the Euclidean algorithm (you may remember from programming languages:

```
int gcd (int a, int b) {
   while (a!=b) {
      if (a>b) a = a-b;
      else b=b-a;
    }
    return a;
}
```

   M where $\mathcal{L}$(M)=RELPRIME
   M = "On input <x, y> where x and y are natural numbers in binary:
   1. Repeat until x=y:
   2. If (x>y) then x = x-y; else y=y-x
   3. If x is 1, accept; else reject."

   Analysis on n, the size of the representation of the 2 numbers. (Recall we are using base 2, so $35 = 100011_2$ which is 6.)
   Step 1. $O(n^2)$ – running back and forth marking seeing if digits the same and marking what was checked.
   Step 2. $O(n^2)$ to see which is larger, running back and forth replacing symbols in x with $, and symbols in y with blank.
   What order of time to do the subtraction? $O(n^?)$
   How many times would we need to execute the loop?
   Step 3, constant time.