CMSC 351: P, NP, Etc. Part 2

Justin Wyss-Gallifent

December 2, 2023

1 Polynomial Time

A reminder:

Definition 1.0.1. An algorithm runs in *polynomial time* if $T(n) = \mathcal{O}(n^k)$ for some $k \in \mathbb{N}$ where n is the input size.

Example 1.1. MergeSort has $T(n) = \Theta(n \lg n) = \mathcal{O}(n^2)$ and hence MergeSort is polynomial time. ■

Example 1.2. Generating a list of all permutations of $\{1, 2, ..., n\}$ has $T(n) = \Theta(n!)$ which is not polynomial time. Why not? Can you prove this? ■

Generally speaking we think of polynomial time as "fast" but depending on the coefficients, in reality polynomial time can be incredibly slow.

2 P

Definition 2.0.1. The set P is the set of all decision problems Q such that $Q \in P$ when there is a DTM polynomial-time algorithm (polynomial in the size of the instance) such that:

- If $Q(I) = YES$ then the algorthm outputs YES.
- If $Q(I) = NO$ then the algorithm outputs NO.

Example 2.1. Q: Given a list A, is A sorted?

We can write a polynomial-time algorithm which takes an instance (a list) A and checks whether each element is greater than or equal to the previous element and returns YES if they all are and NO otherwise.

Thus this decision problem is P.

■

Example 2.2. Q: Given x, y and d, is $d = \gcd(x, y)$?

We can write a polynomial-time algorithm which takes an instance (a triple x, y, d), calculates the gcd of x and y and compares it to d. If equal it returns YES and otherwise NO.

Thus this decision problem is P.

■

Example 2.3. Q: Given two lists A and B of the same length, do they contain the same values?

We can write a polynomial-time algorithm which takes an instance (two lists A and B) sortes them and then compares them element by element. If all elements are equal, return YES, otherwise return NO.

Thus this decision problem is P. ■

Example 2.4. Q: Given a graph G on V vertices and two specific vertices s and t and a distance d, is there a path of length less than or equal to d from s to t?

We can write a polynomial-time algorithm which takes an instance (the data G, V, s, t, d), runs the shortest path algorithm using s as the starting vertex and checks if the distance from s to t is less than d and replies YES or NO accordingly.

Thus this decision problem is P. ■

Example 2.5. Q: Given a partially filled Sudoku board, is there a solution?

There is no known polynomial-time algorithm which takes an instance (a partially filled Sudoku board) and replies YES or NO accordingly.

Thus it is unknown if this decision problem is P. It is suspected that the answer is no.

■

Example 2.6. Q: Given a set A is there a subset which adds to 0?

There is no known polynomial-time algorithm which takes an instance (a set A) and replies YES or NO accordingly.

Thus it is unknown if this decision problem is P. It is suspected that the answer is no.

■

Example 2.7. Q: Given a graph G, does G contain a Hamiltonian cycle? (This is a cycle which contains each vertex exactly once.)

There is no known polynomial-time algorithm which takes any instance (a graph G) and replies YES or NO accordingly.

Thus it is unknown if this decision problem is P. It is suspected that the answer is no. ■

Example 2.8. Q: Given a program which may or may halt, does it halt?

It has been proven that there is no polynomial-time algorithm which takes any instance (a program) and replies YES or NO accordingly.

Thus this decision problem is not in P. ■

In many cases if a decision version of an optimization problem is P then the optimization problem itself can be solved in polynomial time.

Example 2.9. Given a weighted connected graph G and vertices s, t consider the optimization problem of finding the length of the shortest path from s to t.

A decision version of this is:

Q: Given a weighted connected graph T, vertices s, t, and a value k is there a path from s to t of length less than or equal to k?

Suppose we have a polnomial-time algorithm `pathexists(G,s,t,k)` which answers the decision version in polynomial time. In other words within polynomial time it returns YES if there is a path from s to t of length less than or equal to k and NO if not.

Since the graph is connected we know there is a path from s to t and the length of this path could at most be the total sum of all the edge weights. So what we can do is:

```
function findshortestpathlength(G,s,t,k)
    max = sum of edge weights in G
    shortest = max
    for i = max down to 0
        if pathexists(G,s,t,i) then
            shortest = i
        end
    end
    return(shortest)
end
```

This function will return the length of the shortest path and will do so in polynomial time on a DTM.

Thus this optimization process can be solved in polynomial time. ■

3 NP

3.1 Definition

Before talking about NP, here is P again so we can compare:

Definition 3.1.1. The set P is the set of all decision problems Q such that $Q \in P$ when there is a DTM polynomial-time algorithm (polynomial in the size of the instance) such that:

- If $Q(I) = YES$ then the algorthm outputs YES.
- If $Q(I) = NO$ then the algorithm outputs NO.

And now NP:

Definition 3.1.2. The set NP is the set of all decision problems Q such that $Q \in NP$ when there is a NTM polynomial-time algorithm (polynomial in the size of the instance) such that:

- If $Q(I) = YES$ then the algorthm outputs YES.
- If $Q(I) = NO$ then the algorithm outputs NO.

Now then, let's stop to point out that this was the original definition of NP but there's an equivalent and more modern definition of NP which is based upon verification of solutions. Here is the simple version:

Definition 3.1.3. The set NP is the set of all decision problems Q such that $Q \in NP$ when there is a DTM polynomial-time algorithm V called a *verifier* that verifies an instance and potential witness in polynomial time (polynomial in the size of the instance and witness). This means for an instance I and potential witness x that:

- If $V(I, x) = YES$ then there is a witness, although it may not necessarily be x.
- If $V(I, x) = NO$ then x is not a witness.

Note 3.1.1. The idea is that V can use x but doesn't have to. The key is that it must run in polynomial time.

Note 3.1.2. A small warning here that neither bullet point is an iff. This means that if there is a witness we don't necessarily get YES from V and if x is not a witness we don't necessarily get NO from V.

This warning is clarified by the following two examples:

Example 3.1. Q: Given a set of n integers is there a subset which adds to 0?

The verifer $V(I, x)$ takes the set and a particular subset. It sums the values in x and sees if the result is 0 and returns YES or NO accordingly. The following calls yield the following results:

We find $V(\{3,4,2,-7\},\{3,4,-7\}) = YES$ and we can conclude there is a witness.

We find $V(\{3,4,2,-7\},\{3,4\}) = NO$ and we can conclude that $\{3,4\}$ is not a witness. This does not mean there isn't a witness. In this case there is. ■

Example 3.2. Q: Given a list of n integers are any of them greater than 100?

The verifier $V(I,x)$ takes the list and a particular element. It checks the list and sees if there is a value greater than 100 and return YES or NO accordingly. It ignores the particular element. The following calls yield the following results:

We find $V(\{1,5,103,10\},103) = YES$ and we can conclude there is a witness.

We find $V(\{1,5,103,10\},10) = YES$ and we can conclude there is a witness.

We find $V(\{1,5,3,10\},10) = NO$ and we can conclude that 10 is not a witness. This does not mean there isn't a witness. In this case there isn't. ■

Here are some other examples.

Example 3.3. Q: Given a set S of integers can we partition S into two subsets A and B whose sums are equal?

We can write a polynomial-time verifier which takes an instance (a set S) and a potential witness (two subsets A and B), sums the elements in A and B separately, checks if they are equal and replies YES or NO accordingly. This will satisfy the definition.

Thus this decision problem is NP.

Example 3.4. Q: Given a partially-filled sudoku board is there a solution?

We can write a polynomial-time verifer which takes an instance (a partially-filled sudoku board) and a potential witness (a potential solution) and checks if the solution works, replying YES or NO accordingly. This will satisfy the definition.

Thus this decision problem is NP. ■

Some other problems which can be seen to be NP in a similar way:

Example 3.5. Q: Given a set of n integers, is there a subset which adds to 0? ■

| **Example 3.6.** Q: Given a graph G does it contain a cycle? ■

Note 3.1.3. I read somewhere once that some people believe that we should just use VP to mean verifiable in polynomial time on a DTM instead of using NP. That way the machine is always a DTM.

Or even better, SP (for solvable in polynomial time) and VP (verifiable in polynomial time). But there we go.

3.2 An Algorithm for a Verifier which Proves NP

This is an informal presentation of how we might write a verifier which proves a decision problem Q is in NP. The following algorithm will need to run in polynomial time as a function of the sizes of I and x:

```
function V(I,x)
    if we can decide Q(I) without looking at x
        decide Q(I)
        return YES or NO accordingly
    else
        if x is a valid witness?
            return YES
        end if
    end if
    return NO
```

Now pretend that all you know is the input and output of the verifier. Observe that:

- If the verifier returns YES then all you can conclude is that there is a valid witness. You don't know if x is valid because you don't know if the algorithm decided Q or checked x.
- If the verifier returns NO then all you know is that x is not a valid witness. You don't know if there is a valid witness

This algorithm satisfies the definition of a polynomial-time verifier and proves that $Q \in NP$.

4 P v NP

Note 4.0.1. First, observe that $P \subseteq NP$. To see this, note that if a decision problem is in P then if we are given an instance and a witness we can just ignore the witness, run the polynomial-time decider and say YES or NO accordingly.

Definition 4.0.1. The *P v NP Problem* asks whether $P = NP$ or not. In other words is it the case that when we can verify any potential witness in polynomial time that we can also solve the decision problem in polynomial time?

This is perhaps the greatest unsolved problem in computer science. There is overwhelming evidence that $P \neq NP$ in the sense that there are many important problems for which potential witnesses can be verified in polynomial time but no polynomial-time solution has been found. However note that this does not mean that such solutions don't exist.

5 Problem Reduction and Equivalence

5.1 Introduction

This last section looks at what it might mean if solving one decision problem might be "as easy as" solving another. Before diving into this concept we'll first look at a similar idea with sets.

5.2 Using Explicit Sets

First of all observe that a set A can be thought of as a decision problem Q if we treat the elements in A as instances and say that $Q(I) = YES$ if $I \in A$ and $Q(I) = NO$ if $I \notin A$.

Example 5.1. If $A = \{2, 5, 10\}$ then $Q(2) = YES$ and $Q(3) = NO$.

■

Example 5.2. Let A be the set of integer multiple of 3. Then $Q(6) = YES$ and $Q(5) = NO$.

■

Example 5.3. Let A be the set of partially filled sudoku boards which are solvable. Clearly there are some partially filled boards x for which $Q(x) = TRUE$ and some for which $Q(x) = FALSE$.

■

Definition 5.2.1. Given sets A and B we say that A is *polynomially reducible* to B if there is some function p which can be computed in polynomial time and such that $x \in A$ iff $p(x) \in B$.

Note 5.2.1. The idea here is that making a decision about whether $x \in A$ or not can be, in polynomial time, altered to a question about whether $p(x) \in B$ or not.

Example 5.4. Let A be the set of integer multiples of 2 and let B be the set of integer multiples of 3.

To prove that A is polynomially reducible to B we define p by $p(x) = 3x/2$. Integer multiplication is a polynomial-time procedure.

Observe that:

$\Longrightarrow$: If $x \in A$ then $x = 2k$ for some $k \in \mathbb{Z}$ and then $p(x) = 3x/2 = 3(2k)/2 = 3k$ so $p(x) \in B$.

$\Longleftarrow$: If $p(x) \in B$ then $p(x) = 3k$ for some $k \in \mathbb{Z}$ and then $3x/2 = 3k$ so $x = 2k$ so $x \in A$.

■

Example 5.5. Define the sets A and B as follows:

$$A = \{s \big| s \text{ is a string}\}$$

$$B = \{s \big| s \text{ is a string ending with "Z"}\}$$

To prove that A is polynomially reducible to B we define p by $p(s) = s + \text{``}Z''$ where + is string concatenation. String concatenation is a polynomial-time procedure.

Observe that:

$\Longrightarrow$: If $s \in A$ then $p(s) = s +'' Z''$ ends with "Z'' so $p(x) \in B$.

$\Longleftarrow$: If $p(s) \in B$ then $p(s)$ ends with "Z''. Thus $s +'' Z''$ ends with "Z" and so s is a string so $s \in A$.

■

5.3 Decision Problems and Algorithms

Now let's work this up to some algorithms. First, here's the definition:

Definition 5.3.1. For decision problems Q_1 and Q_2 We say that Q_1 is *polynomially reducible* to Q_2 if there is some algorithm p which runs in polynomial time which convertes instances for Q_1 to instances for Q_2 such that $Q_1(I) = YES$ iff $Q_2(p(I)) = YES$.

If this is the case then we'll write:

$$Q_1 \leq_P Q_2$$

Note 5.3.1. It's not important how long Q_2 takes and sometimes Q_2 is called an *oracle* instead to try to impart the idea that it just knows with no work at all.

Example 5.6. Suppose a function `ORACLE(n)` decides something and returns `YES` or `NO`. Consider the following algorithm for `QUESTION(n)`:

```
function QUESTIONS(n)
    for i = 1 to n
        if ORACLE(i)
            return(YES)
        end
    end
    return(NO)
end
```

Observe that `QUESTION` is polynomially reducible to `ORACLE`. We would thus write $QUESTION \leq_P ORACLE$.

Note that there might be other algorithms that do whatever `QUESTION` does and they may do it faster, but we don't know. What we do know, however, is that this algorithm for `QUESTION` reduces the problem to `ORACLE` in polynomial time so it's essentially "no harder than" `ORACLE(n)`. ■

■

Here is another example:

Example 5.7. Consider these two decision problems:

$ISHAMILTON(G)$: Given an undirected graph on n vertices, is there a Hamiltonian cycle in the graph?

$ORACLE(G)$: Given a directed graph on n vertices, is there a Hamiltonian cycle in the graph?

Given a undirected graph G the adjency matrix for G also represents the adjency matrix for G' where G' is obtained from G by replacing each (undirected) edge with two edges, one in each direction. This takes no time. We can then apply `ORACLE(G)` to find a Hamiltonian cycle in G' which is also a Hamiltonial Cycle in G.

It follows that $ISHAMILTON \leq_P ORACLE$.

Now then, it is widely believed that $ISHAMILTON \notin P$ and so if this is true, then $ORACLE \notin P$ also.

■

And one more:

Example 5.8. Consider these two decision problems:

$ISZEROSUBSET(S)$: Given a set S of n integers is there a subset which adds to 0?

$ORACLE(S, x)$: Given a set S of n integers and one integer x in the set, is there a subset of $S - \{x\}$ which adds to $-x$?

To see this suppose we have a set S of integers. We iterate through each element x of S and for each we ask if $ORACLE(S, x) == YES$. If it is true for at least one x then we return YES for $ISZEROSUBSET(S)$ and otherwise we return NO.

It follows that $ISZEROSUBSET \leq_P ORACLE$.

■

5.4 Stepping Away from Decision Problems

This same idea of polynomial reduction can then apply to problems which are not decision problems.

> **Example 5.9.** Consider the non-decision problem:
>
> $SOLVE(B)$: Given a partially filled $n^2 \times n^2$ sudoku board B which has a solution, find it.
>
> And the decision problem:
>
> $ORACLE(B, x, y, v)$: Given a partially filled $n^2 \times n^2$ sudoku board B, is there a solution with the value v placed into the empty space with coordinates (x, y)?
>
> Observe that if we are given a partially filled sudoku board we can iterate through the empty spaces and for each one we can test values 1 through n^2 using $ORACLE$. We know there's a solution so for each empty space we will find a value which works so we add this to our solution. At the end we have solved it. Note that there are at fewer than $(n^2)(n^2) = n^4$ empty spaces and we have to test at most n^2 values in each space so this is $\mathcal{O}(n^6)$.
>
> Thus we can calculate $SOLVE$ in polynomial time as a function of $ORACLE$.

5.5 Some Facts about Polynomial Reducibility

Now we can formalize some facts for decision problems Q_1 and Q_2:

- If $Q_1 \leq_P Q_2$ and $Q_2 \in P$ then $Q_1 \in P$.
- If $Q_1 \leq_P Q_2$ and $Q_1 \notin P$ then $Q_2 \notin P$.

For the mathematicians here, a nice way of thinking about this is to imagine three functions $q_1(x)$, $q_2(x)$, and $p(x)$. Suppose we know for sure that $p(x)$ is a polynomial and we know that $q_1(x) = p(q_2(x))$.

Then we can say:

- q_1 is a polynomial in terms of q_1.
- If $q_2(x)$ is a polynomial then $q_1(x)$ is a polynomial.
- If $q_1(x)$ is not a polynomial then $q_2(x)$ is not a polynomial.

Note that if $p(x)$ is not a polynomial we can say nothing.

6 Thoughts, Problems, Ideas

1. Explain how you know that the following decision problems are in P. You don't need to provide pseucode, a basic explanation will suffice.

   (a) $\mathrm{Y} \vee \mathrm{N}$: Given a list with n elements, is it unsorted?

   (b) $\mathrm{Y} \vee \mathrm{N}$: Given the adjacency matrix for a graph with n vertices, is there one vertex which is connected to all the others?

   (c) $\mathrm{Y} \vee \mathrm{N}$: Given base-10 list representations of two n digit numbers A and B, is $AB \geq 5 \cdot 10^{2n-2}$?
   For example is $84 \cdot 23 \geq 58 \cdot 10^2 = 500$?

   (d) $\mathrm{Y} \vee \mathrm{N}$: Given a list with n elements, is the maximum to the left of the minimum?

2. For each of the following you are given a problem $PROB$ and an associated decision problem DEC. For each, write pseudocode to show that if $DEC \in P$ then $PROB$ can be solved in polynomial time.
   Note: Don't worry about whether or not it's true in the real world that $DEC \in P$, just assume it is and base your pseudocode on it.

   (a) Given a simple connected unweighted graph with n vertices.
   $PROB$: Find length of the longest path.
   DEC: For any given k, is there a path of length k?

   (b) Given a list A of n integers, a subset $S \subset A$, and a target t.
   $PROB$: Assuming there is a subset of A containing S which sums to t, find it.
   DEC: Is there a subset of A containing S which sums to t?

   (c) Given an integer $n \geq 2$.
   $PROB$: Find the smallest prime factor of n.
   $PROB$: For any given k, is k prime?

3. Explain why reverse-sorting a list is polynomially reducible to sorting a list.

4. Suppose a garage contains n motorcycles each of which has 1, 2 or 4 cylinders. Explain why fixing all cylinders on all motorcycles is polynomially reducible to fixing one cylinder.

5. Suppose G is a simple graph with n vertices. Explain why counting the edges in the graph is polnomially reducible to calculating the degree of a vertex.