THE P VS. NP PROBLEM

GLENN SUN
OLGA RADKO MATH CIRCLE ADVANCED 2

NOVEMBER 22, 2020

Last week, we studied an introduction to algorithms, and we learned about analyzing the running time of algorithms with big-O notation. Today, we'll apply those ideas to learning about the P vs. NP problem, one of the most important open problems in mathematics.

Definition 1 (decision problem)**.** A decision problem is a question where the answer is always either yes or no.

Exercise 1. Last week, we started by discussing four problems. They are restated here. Which of them are decision problems?

1. Given a list of numbers $(x_1, \ldots, x_n)$, find the biggest number.
2. Given a list of numbers $(x_1, \ldots, x_n)$, find the range of the list, that is, the maximum value minus the minimum value.
3. Given two lists of numbers $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$, determine if the two lists have any numbers in common.
4. Given a list of numbers $(x_1, \ldots, x_n)$, sort the list from smallest to biggest.

Definition 2. P is the class of decision problems that can be solved with an algorithm that runs in polynomial time, in other words, $O(n^k)$ time for some constant k.

Exercise 2. Show that the following decision problems are in P by designing an algorithm that solves them in polynomial time.

1. Given a list of numbers $(x_1, \ldots, x_n)$, determine if any number is repeated.
2. Given a set of points in the plane $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, determine if there exists three points that form an equilateral triangle.

Now let us turn out attention to NP. Here is a motivating example.

Exercise 3. The subset sum problem is defined as follows: Given a set of positive integers S and another integer k, determine if there exists a subset $S' \subseteq S$ that you can add up to get exactly k.

In other words, (S, k) is a yes-instance (meaning the answer should be yes) if and only if there exists a subset $S' \subseteq S$ that you can add up to get exactly k.

In this question, we're *not* interested determining whether (S, k) is a yes-instance or no-instace. Instead write a polynomial time algorithm that takes (S, k, S') as input and satisfies the following properties:

- If (S, k) is a yes-instance, then there exists S' such that the algorithm returns "yes" when given (S, k, S') as input.
- If (S, k) is a no-instance, then for all S', the algorithm returns "no" when given (S, k, S') as input.

In other words, you just showed that subset sum can be checked in polynomial time, as long as you are provided a *certificate*. In the above case, the certificate was the subset that adds up to exactly k. In general, NP is the class of problems that can be *checked* in polynomial time, and we formalize the notion of checking as follows.

Definition 3. Consider a decision problem and call the input x. To say that the decision problem is in NP, we require that there exists a polynomial time algorithm, called a *verifier*, that takes (x, u) as input and does the following:

- If x is a yes-instance, the algorithm returns "yes" for some certificate u.
- If x is a no-instance, the algorithm returns "no" for all certificates u.

Sometimes, the certificate is also called a proof. The relationship with mathematical proofs is the following:

- If you are given a true statement, then for some proof of the statement, you can read it and verify that the statement is true.
- If you are given a false statement, then for all supposed proofs of the statement, you can find a flaw and be not convinced.

This is exactly what the verifier does for an NP problem.

Contrary to popular belief, the N in NP stands for "nondeterministic", not just "non". The sense in which NP has to do with nondeterminism is an advanced topic that is beyond the scope of today's discussion. But this certificate-verifier definition above will be sufficient for our purposes today.

Exercise 4. Show that the following decision problems are in NP by finding a suitable certificate that can be verified in polynomial time.

1. Given a set of objects X, a bunch of subsets $S_1, \ldots, S_n$ of X, and an integer k, determine if you can hit every subset by picking at most k objects from X. (This is called the hitting set problem.)
2. Given a set of objects of potentially different weights and values, and a bag that can hold at most N units of weight, determine if you can fill the bag with objects having total value at least k. (This is called the knapsack problem.)

Exercise 5. Explain why if a problem is in P, then it must also be in NP.

The P vs. NP conjecture simply says that the converse is false. That is, there are problems that can be checked in polynomial time, but cannot be solved directly in polynomial time.

Exercise 6. Write an algorithm to solve the subset sum problem. Your algorithm should not run in polynomial time, unless you want to earn $1 million right now.

Exercise 7. Try to come up with reasons why you think the subset sum problem cannot be solved in polynomial time. Most likely something about your reasoning will be wrong, so have your instructor or other students find the hole in your reasoning. You can stop when you are convinced that this is a difficult question.

The reason why the previous two exercises asked about the subset sum problem is because the subset sum problem is what is called an NP-complete problem. You can think of NP-complete problems as the most difficult problems in NP.

Definition 4. A decision problem is NP-complete if the following two things are true:

1. The problem is in NP, and
2. If this problem could be solved in polynomial time, then any problem in NP can be solved in polynomial time.

It's amazing that NP-complete problems exist at all, but Stephen Cook and Leonid Levin proved in 1971 that they do exist, and in fact they found dozens of interesting NP-complete problems. It turns out that subset sum is one of these NP-complete problems.

NP-complete problems are relevant to the P vs. NP conjecture because if you can find a polynomial time algorithm to any NP-complete problem, it would disprove the conjecture, since it would imply that any problem in NP is solvable in polynomial time.

Exercise 8. If you found a polynomial time algorithm to an NP problem that was not NP-complete, would it resolve the question of whether or not P is equal to NP?

Exercise 9 (Challenge)**.** Given a set of positive integers S, determine if S can be split into two subsets that have equal sum. This is called the partition problem. Using the fact that subset sum is NP-complete, prove that the partition problem is also NP-complete. (This kind of argument is called a *reduction*.)

1. Prove that the partition problem is in NP.
2. Show that if partition problem can be solved in polynomial time, then the subset sum problem can be solved in polynomial time. Here are the steps to take:
   (a) Let a denote the sum of the elements in S. Show that $S \cup \{a - 2k\}$ (the disjoint union) can be split into two subsets of equal sum if and only if S has a subset that sums to k.

(b) Use the above fact to solve subset sum using partition.

3. Conclude that the partition problem is NP-complete.

There are now hundreds, or perhaps thousands, of known NP-complete problems. People have even proved that generalized versions of games like Battleship and Sudoku are NP-complete!