❖ NP-Completeness

In this final lecture, we will discuss some results around computational complexity and its implications on our lives as professionals working in computing.

9.1 Reductions

Oftentimes problems that seem new or difficult to solve can be **reduced** to instances of problems we know how to solve. Consider the **arbitrage problem**. An arbitrage opportunity in a market is when exchange rates are in a state where there exists a sequence of exchanges to increase your money.

Question 148. Suppose a is the exchange rate to go from dollars to yen, b is the exchange rate to go from yen to euros, and c is the exchange rate to go from euros to dollars. When is there an arbitrage opportunity in a graph?

We have found a way to discover a condition regarding cycles which tells us when an arbitrage opportunity exists. We know another result related to detecting cycles, can we bridge these two somehow?

Question 149. What is a property of a cycle that we know how to detect? How can we relate this to an arbitrage opportunity?

9.2 Decision vs Search Problems

Computer scientists are interested in classifying problems into families, based on how difficult we think they are to solve. The main objects we will study for this are called **decision problems**. A decision problem is a problem whose output is either 0 or 1, in contrast to a **search problem** where we are looking for a particular answer. An example that distinguishes these two can be seen in something like Sudoku.

9	1	3				5		
6		7					2	4
	5			8			7	
	7	9						
		2		9			4	3
					4		9	
	4				1	9		
7		6			9			5
		1			6	4		7

The regular Sudoku puzzle like the above is a search problem, because you have to find an exact assignment of numbers to the empty cells to satisfy the solution. In contrast with this, we could ask a decision problem version of sudoku, where you just have to YES if the sudoku puzzle can be solved, and NO otherwise.

Question 150. Suppose we could talk to an all knowing oracle who could answer the decision version of Sudoku. Is there a way to use this oracle to efficiently solve the search version?

For most problems we are interested in, it is sufficient to solve the decision problem to solve the search problems as well. Because of this computer scientists isolate the study of how hard a problem is to the decision version. We cluster problems into many different classes, and most problems we looked at in this class fall into the class P. P stands for "deterministic **P**olynomial time", with the emphasis on polynomial. It is basically the set of all problems that can be solved using a polynomial time algorithm.

NP stands for **non-deterministic polynomial time**. We won't spend time defining this in this class (if you are curious, you can learn more about it in CS162), but here is the "English" definition of the family. We will say a problem is in NP if

- When the answer is YES, there is an easy proof that will convince that is the case.
- When the answer is NO, no proof will convince me that the answer is YES.

Question 151. Show that the problem Sudoku is in NP.

Question 152. Draw the relation between the sets P and NP.

Some problems that we saw in class also fall into the class NP. Let's try to reason about the 0-1 Knapsack problem, but again we will focus on the decision version of it. We summarize the decision problem as follows.

Problem (0-1 Knapsack Decision)**.**

INPUT: A list of n items with their weights and values, and a knapsack that can hold up to W kilograms, and a target value T.

OUTPUT: YES if there is a way to load some of the items in the knapsack without exceeding W kilograms, but getting a value of T.

Question 153. If the answer is YES, how can someone convince you that the answer is YES?

Question 154. If the answer is NO, why would you never believe what they claim?

Question 155. If we had access to an oracle who could instantly solve the decision version of 0-1 knapsack, how could we use it to solve the search version?

9.3 NP-completeness

A problem is NP-complete, if

1. the problem is in NP, and
2. every problem in NP can be reduced to it.

We now know many different NP-complete problems, and they include many important problems that people are interested in, some of which may affect the security of our online banks, maximize the efficiency of delivery systems, and immediately prove any mathematical statement (that can be proven). Furthermore, if someone can solve **one** NP-complete problem, this means we can solve **ALL** NP-complete problems. Right now, the best algorithm that we have for any NP-complete problem takes exponential time.

Question 156. What was the runtime of 0-1 knapsack?

Question 157. What happens to the runtime of 0-1 knapsack if we double the number of digits representing the weight of the knapsack, how does the runtime change?

We showed that 0-1 knapsack is NP-complete, and it turns out that generalized sudoku (sudoku on an n by n grid) is also NP-complete. What this means is that if you can find a polynomial time algorithm to solve either of them, you can solve the other one! These unexpected connections span a huge zoo of problems, that there is even a Wikipedia page that lists some representative NP-complete problems, of which there are over a 1000 of that we know by now.

No one has been able to prove that NP-complete problems have a polynomial time algorithm, but no one has been able to prove that there **doesn't** exist any either. This means that there is a possibility that a polynomial time algorithm for them exists, but the general consensus between computer scientists today is that this is unlikely to be the case. There is a million dollar prize reserved for anyone who can definitively prove whether this is the case, but progress is quite slow.

Due to this, I like to describe the gap between P and NP almost like a physical constant of the universe, similar to the way that we have a "guess" for what the speed of light is, and can build many theories on top of this belief.

A common illusion that the tech industry uses to garner funds and public interest is to promise that their product will somehow be able to solve problems that are supposed to be NP-complete. Recently this is used a lot by AI developers, claiming that they were able to use AI to solve some problem which is supposed to be NP-complete. If you think this is too good to be true, it probably is, and oftentimes the technology fails.

As you venture out to the world as professionals equipped with computer science knowledge, I hope you are able to discern when this lie is being used and identify the NP-complete problem hiding beneath the promises. Furthermore, I hope we all use our knowledge in a constructive way, to acknowledge the shortcomings we have and to use this knowledge as a way to further appreciate the successes we achieve.