Notes 8.370/18.435 Fall 2022

Lecture 18 Prof. Peter Shor

Today, I'm going to talk about classical complexity theory, and then I'll talk about oracles. I'll explain some ways in which oracles are used in classical theoretical computer science, and then talk about how quantum oracles work.

Theoretical computer scientists loosely consider an algorithm thar runs in time that is a polynomial in the length n of its input as "efficient" and an algorithm that runs in time super-polynomial in the length of its input as "inefficient." Here, what running time means is simply the number of elementary steps the program takes. This is clearly wrong, in terms of actual intuitive notions of efficient and inefficient. An algorithm that takes time n^{24} will only run in a reasonable time for very small input sizes n, and there are several algorithms, like the simplex algorithm, which have a worst-case running time that is exponential in the input size, but which actually run fast nearly all the time.

So how did this non-intuitive state of affairs develop?

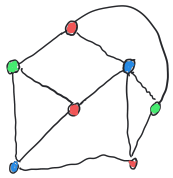
Back in the early days of computing, there were many different kinds of models of computational machines. The first mathematical model of a computing machine developed was put forward by Alan Turing, in 1936, a decade or so before the advent of actual computers. This is a *Turing machine*, and it is still one of the main models used in classes on complexity theory. Another model is a random-access machine (RAM), which is closer to what a real-life computer looks like. When you're talking about abstract models of computation, you'd like to define complexity classes that are independent of the exact details of machine you're running on. But an algorithm that runs in n time on a RAM might take n^2 time on a Turing machine. So several computer scientists, most notably Alan Cobham, decided to define the complexity class P of problems which could be solved with an algorithm that ran in polynomial time, The difference between the different computational models was in all cases that these computer scientists considered only a polynomial factor, so this difference was not relevant for the question of whether a problem is in P.

There is a larger class than P, namely BPP.[1] This is the set of problems that can be solved with high probability with a randomized algorithm that runs in polynomial time. For quantum complexity classes, the most relevant complexity class is BQP, the class of problems that can be solved on a quantum computer with high probability in polynomial time. Here, because quantum computation is inherently probabilistic, the most natural class is randomized; while people have defined deterministic classes, these classes appear to depend on the exact details of the definition, and are thus less natural.

We now want to talk about the complexity class NP. This is the set of problems that can be easily verified. An example is three-colorability. A graph is three-colorable if its vertices can be colored so that no edge connects two vertices of the same color (see the figure below for an example).

[1] although it has not been proved larger yet, and some theoretical computer scientists think it's the same

Figure: A 3-colorable graph.



If you're given a graph, there is no polynomial time algorithm known that will tell whether this graph is three-colorable. However, if you're given a graph and a coloring, it is easy to check whether the coloring is a three-coloring of the graph.

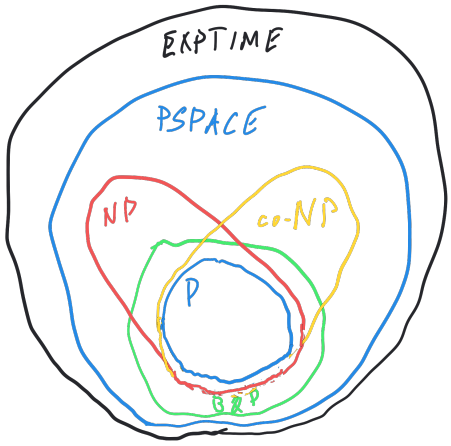
Formally, a problem is in NP if there is a program that runs in polynomial time, which takes as input the problem instance and another input called the "witness", and outputs "accept" or "reject". For a problem instance that does not have the property (being three-colorable in our example), this program must always output "reject", and for one that does, there must always be some witness that will make the program output "accept". For instance, for three-colorability, you would input the graph (the problem instance) and a three-coloring of the graph (the witness); the program would check that there are not edges with both endpoints colored by the same color, and output "accept" if this holds.

Some problems in NP, (three-colorability is one) are NP-complete. We can prove that if you can solve one of these problems in polynomial time, you can solve all problems in NP in polynomial time. How can you do that? First, you need to find a master problem that is NP-complete. The most commonly used problem for this is 3-SAT. I'm not going into the details, but you can express any computation as a Boolean formula, and you can turn any Boolean formula into a 3-SAT problem. So this shows that any problem in NP can be turned into a 3-SAT problem.

Now, to show that 3-colorability is NP-complete, you need to show that for any 3-SAT formula, you can find a graph that's only larger by a polynomial factor, and for which the graph is 3-colorable if and only if the 3-SAT formula is satisfiable.

So where does BQP fall with respect to the complexity class NP? It is not believed that NP is contained in BQP, but we also do not believe that quantum computers can solve NP-complete problems in polynomial time. Thus, the general belief among theoretical computer scientists is that they are incomparable. This can be seen in the following Venn diagram. Here, co-NP is the complement language of NP: problems for which a negative solution can be verified. PSPACE is the set of problems that have algorithms which will solve them with polynomial space, or memory, rather than time. And EXPTIME is the set of problems that can be solved in time exponential in their input size. It turns out that quantum PSPACE is the same as PSPACE, so quantum computing cannot reduce the amount of space needed to solve a problem by more than a polynomial factor. The only two complexity classes in the diagram which theoretical computer scientists can prove are different are P and EXPTIME, which shows how hard it is to prove that two complexity classes are different.

The rest of this lecture was spent introducing the quantum Fourier transform, and I will write this part up in the lecture notes for Wednesday's lecture.


EXPTIME
PSPACE
NP
co-NP
P
BPP