Lecture 19: P and NP, and The Big Question I

built on 2021/03/09 at 09:39:25

1 Some Easy, Some Hard

Some tasks in computer science are known to be easy: given a sequence of n numbers, we know how to sort them from small to large quickly. There are many algorithms that achieve $O(n \log n)$ time.

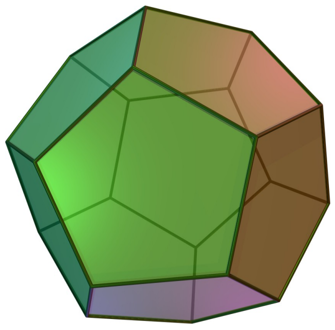
Consider another task: given an undirected graph G, find a way to walk from a starting point going through each and every edge once and returning to the starting point. This is the famous *Eulerian cycle* problem, which can be solved in $O(n + m)$ time. Here n is the number of vertices and m is the number of edges in the given graph. This rests on a theorem from graph theory, which isn't difficult to prove.

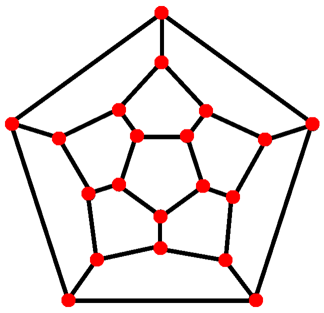
Theorem 1.1 *A connected graph has an Eulerian cycle if and only if every vertex has an even degree.*

Using this theorem, it is easy to determine which of the following graphs has an Eulerian cycle—and find such a cycle when it exists.

```
  o       ..o
 / \     /   \
o---o   / o---o
|\ /|  o  |  /|
| X |   \ | / |
|/ \|    \|/  |
o---o     o---o
```

Consider yet another task: given an undirected graph G, find a way to walk around the graph while visiting each vertex once and only once—and come back to where we start. A version of this problem was proposed by an English astronomer Sir William Rowan Hamilton in the form of a puzzle. This is why such a walk is known as a *Hamiltonian cycle*. To see why this problem is interesting, take a dodecahedron: if we make a graph out of it, there are 20 vertices, 30 edges, and 12 faces.

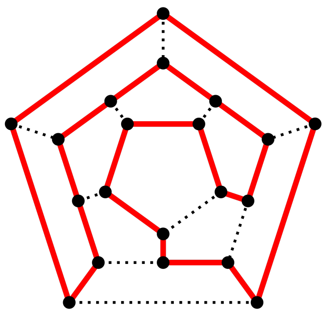




(Figures from Wikipedia)

Does this graph have a Hamiltonian cycle? We'll try our hands at it for a while. Do we have any strategy for coming up with a solution? At first, this puzzle doesn't seem much different from the Eulerian cycle problem: Eulerian uses each edge exactly once whereas Hamiltonian uses each vertex exactly once. But as far as we know, surprisingly, there is no simple property that tells apart graphs that have Hamiltonian cycle from graphs that don't.

In the particular case of a dodecahedron, it turns out that there is such a cycle (see one such cycle in the figure below).



The process of coming up with a cycle like this, however, is nothing more glamorous than exhaustively trying all possible cycles. **Is Hamiltonian cycle fundamentally harder than the easier ones we've studied—or we haven't been smart enough to find a clever solution?**

2 The NP Problems

As it turns out, the Hamiltonian cycle problem has a few interesting properties. In fact, it is a typical example of a large number of useful problems out there.

- First, once you have a solution (or claim to have a solution), it is easy to check whether that solution is valid. In the case of Hamiltonian cycle, once someone provides you a walk, you can easily check if it is a cycle that visits every vertex exactly one.
- Second, it is *the process of coming up with a solution* that is difficult. Indeed, as far as we know, the only way to find a solution is to exhaustively search through the possibilities, of which there are too many to be efficient.

For today's lecture, we'll focus on decision problems, problems whose answers are either yes or no. We'll create a class of problems that have essentially this basic logical structure: if there is a solution (and you have that solution), it is easy to verify.

We'll formalize this idea a bit more. First, notice that a decision problem P can be seen as a language. The corresponse is simple: an input x is in this language, say L, if and only if $P(x)$ says "yes." Given this correspondence, we'll abuse notation and use NP to be a class of problems (and technically, equivalently a class of languages).

Definition 2.1 (The Class NP) *A language (or equivalently, a decision problem) $L \in$ NP if and only if there exists an algorithm called the "verifier" such that*

- *V takes two inputs x and y, $|y| \leqslant |x|^{O(1)}$;*
- *$V(x, y)$ runs in $(|x| + |y|)^{O(1)}$ time; and*
- *for all $x \in L$, there exists a y such that $V(x, y)$ says "yes."—whereas for all $x \notin L$, for all y, $V(x, y)$ says "no."*

Remarks: The input x is called the real input—and the input y is called a *witness* or a *certificate*. The notation $n^{O(1)}$ is just a fancy way to write n^{constant}.

By contrast, the class P is the set of all languages such that it is possible to decide whether an input x belongs to that language in $|x|^{O(1)}$ time. That is, given an input, you can decide it in polynomial time.

2.1 What's In NP?

To show that a problem is in NP, we have to show two things:

- For every "yes" input, there is a *polynomially-bounded certificate*.
- An algorithm that runs in polynomial time (poly in the length of the real input and the certificate) such that
  - for every "yes" input, it takes that input and a certificate—and correctly verifies that it is indeed a yes input.

  – for every "no" input, it takes that input and any certificate at all—and correctly rejects it.

Using this definition, let us show that the Hamiltonian cycle problem belongs in NP. From now on, we'll denote the Hamiltonian cycle problem by HAM-CYCLE.

- First, the certificate: if it is a yes instance, the certificate will simply be a list of vertices describing the order in which we visit them. The length of such a certificate is the same as the number of vertices.
- Then, using this certificate, we can just check if we can really walk along the given route. And finally, check if every vertex is visited exactly once. It won't be hard to prove that if there is such a cycle, this verifier will check and accept it. And if there isn't such a cycle, no certificates can cause the verifier to accept. You can convince yourself that this algorithm runs in at most $O(n^2)$ time, which is polynomial in both inputs.

What else is in NP? Coloring. A graph G is k-colorable if it is possible to assign at most k distinct colors to the vertices such that for every edge, the endpoints have different colors.

k-Coloring: Given a graph G as an input and a number k, is the graph k-colorable?

Let's go through the same exercise to show that it belongs in NP. What's your certificate? What's your verification algorithm?

2.2 Reduction

This is a good point to bring back the idea of *reductions*. We have seen reductions in the context of decidability. In that context, we didn't care about efficiency. The notion of reduction we're going to be using will have some mention of efficiency—as we will embark on a journey to tell apart problems that are efficiently solved and problems that can't be efficiently solved.

Polynomial-Time Reduction: A problem A reduces to B in polynomial time, written $A \leqslant_p B$, if for any input I for A, you can convert I to I', which can be solved by B, whose solution $B(I')$ can then be used to provide the answer for $A(I)$. Furthermore, the process of converting I to I'—and converting the answer back—takes polynomial in the input size.

(Draw a diagram for this).

This means:

- If $A \leqslant_p B$, then A is no harder than B, up to polynomial. Furthermore, if B is polynomially solvable, then A is also polynomially solvable.
- If $A \leqslant_p B$ and $B \leqslant_p C$, then we can show that $A \leqslant_p C$.

2.3 The Bread and Butter: Satisfiability

Consider the following Boolean formula:

$$p(x, y, z) = (x \vee \neg y) \wedge (\neg x \vee \neg z) \wedge (y \vee z) \wedge (\neg y \vee z)$$

This formula has a special structure. This is known as the conjunctive normal form (CNF). Here, the formula is made up of several clauses; the clauses are joined by $\wedge$, and each clause uses $\wedge$ to join together literals. In other words, this is ANDs of OR clauses.

Satisfiability(SAT). Given a Boolean formula ϕ, is there a truth assignment so that ϕ evaluates to `true`?

Of course, if you're willing to try all truth assignment—essentially writing out the truth table—you can find the answer. However, this doesn't seem to be very efficient. If the formula has n variables, you will need to try 2^n combinations.

Just like Hamiltonian cycle, the SAT problem is in NP. Can you show that it is?

2.4 $\mathsf{P} \subseteq \mathsf{NP}$

We claim that $\mathsf{P} \subseteq \mathsf{NP}$. The proof is pretty simple. If a problem P is in P, then there is an algorithm P_A that decides any input x in polynomial time (polynomial in $|x|$). Now we'll show that P also belongs in NP. The certificate can be empty; we don't need one. We'll use P_A as the "verifier" algorithm. Indeed, this runs in time polynomial in the input length.

Though $\mathsf{P} \subseteq \mathsf{NP}$ is easy to prove, one of the most famous open-problems in computer science asks whether $\mathsf{P} = \mathsf{NP}$—that is, whether there is anything that is in NP but not in P. We don't yet know the answer, but many experts believe that in fact $\mathsf{P} \neq \mathsf{NP}$.

2.5 The Complement of NP

Having looked at many problems, you may now wonder whether there is anything that doesn't belong in NP.

Let's talk about Ham-Cycle again. Recall that Ham-Cycle asks whether there is a cycle that visits each and every vertex of a graph once. In general, an NP problem can be alternatively viewed as follows: a problem $A \in \mathsf{NP}$ computes

$$A(x) = \exists y.\, V(x, y)$$

where y and the running time of V have to have certain properties.

Consider the complement of a problem A. Briefly,

$$\overline{A(x)} = \forall y.\, \neg V(x, y)$$

In the case of Ham-Cycle, the complement of the problem, which we'll denote by $\overline{\text{Ham-Cycle}}$, is "is it the case that G has no Hamiltonian cycle?"

This turns out to be a different problem. Having a Hamiltonian cycle in hand doesn't do much. To convince someone that $\overline{\text{Ham-Cycle}}$ is a "no" is easy: it's the same old NP. But to convince someone that $\overline{\text{Ham-Cycle}}$ is a "yes" apparently requires testing all possibilities.

$\overline{\text{Ham-Cycle}}$ is an example of a problem in the complexity class coNP. Formally, a problem $A \in \mathsf{coNP}$ if and only if $\overline{A} \in \mathsf{NP}$. Whereas a "yes" input for NP has a simple proof (though a "no" input may be difficult to show), a "no" input for coNP has a simple proof (though, perhaps, a "yes" input may be difficult).

3 Why do you call it NP?

Whereas P stands for the "polynomial time" class, the NP stands for the "nondeterministic polynomial time" class. It's often mistakenly referred to as "nonpolynomial time," which is historically inaccurate.

Suppose your Turing machine has nondeterminism capability (imagine NFA on steroids). This means at each transition, it can be presented with multiple options. When we discussed NFA, we looked at a guesser-verifier view, where someone chooses which option the machine would take at each step, and at the end, it is checked whether all these choices would lead up to an accepting state. A similar model can be defined in the TM setting. The choices you make along the way will be your certificate, and then the verifier checks if the decisions made are right. NP represents what a TM with nondeterminism is capable of computing.

Exercises

1. Prove that k-Coloring $\leqslant_p$ $(k+1)$-Coloring.
2. Prove that 5-Coloring $\leqslant_p$ 4-Coloring.
3. Argue that $\mathsf{P} \subseteq \mathsf{coNP}$, and hence $\mathsf{P} \subseteq \mathsf{NP} \cap \mathsf{coNP}$.