you've heard them all. – jason Jul 10 '13 at 13:55

10 Answers

1256

✓

I assume that you are looking for intuitive definitions, since the technical definitions require quite some time to understand. First of all, let's remember a preliminary needed concept to understand those definitions.

- **Decision problem**: A problem with a **yes** or **no** answer.

Now, let us define those *complexity classes*.

P

P is a complexity class that represents the set of all decision problems that can be solved in polynomial time.

That is, given an instance of the problem, the answer yes or no can be decided in polynomial time.

Example

Given a connected graph `G`, can its vertices be coloured using two colours so that no edge is monochromatic?

Algorithm: start with an arbitrary vertex, color it red and all of its neighbours blue and continue. Stop when you run out of vertices or you are forced to make an edge have both of its endpoints be the same color.

NP

NP is a complexity class that represents the set of all decision problems for which the instances where the answer is "yes" have proofs that can be verified in polynomial time.

This means that if someone gives us an instance of the problem and a certificate (sometimes called a witness) to the answer being yes, we can check that it is correct in polynomial time.

Example

Integer factorisation is in NP. This is the problem that given integers `n` and `m`, is there an integer `f` with `1 < f < m`, such that `f` divides `n` (`f` is a small factor of `n`)?

This is a decision problem because the answers are yes or no. If someone hands us an instance of the problem (so they hand us integers `n` and `m`) and an integer `f` with `1 < f < m`, and claim that `f` is a factor of `n` (the certificate), we can check the answer in *polynomial time* by performing the division `n / f`.

NP-Complete

NP-Complete is a complexity class which represents the set of all problems `X` in NP for which it is possible to reduce any other NP problem `Y` to `X` in polynomial time.

Intuitively this means that we can solve `Y` quickly if we know how to solve `X` quickly. Precisely, `Y` is reducible to `X`, if there is a polynomial time algorithm `f` to transform instances `y` of `Y` to instances `x = f(y)` of `X` in polynomial time, with the property that the answer to `y` is yes, if and only if the answer to `f(y)` is yes.

Example

`3-SAT`. This is the problem wherein we are given a conjunction (ANDs) of 3-clause disjunctions (ORs), statements of the form

```
(x_v11 OR x_v21 OR x_v31) AND
(x_v12 OR x_v22 OR x_v32) AND
...                       AND
(x_v1n OR x_v2n OR x_v3n)
```

where each `x_vij` is a boolean variable or the negation of a variable from a finite predefined list `(x_1, x_2, ... x_n)`.

It can be shown that *every NP problem can be reduced to 3-SAT*. The proof of this is technical and requires use of the technical definition of NP (*based on non-deterministic Turing machines*). This is known as *Cook's theorem*.

What makes NP-complete problems important is that if a deterministic polynomial time algorithm can be found to solve one of them, every NP problem is solvable in polynomial time (one problem to rule them all).

NP-hard

Intuitively, these are the problems that are *at least as hard as the NP-complete problems*. Note that NP-hard problems *do not have to be in NP*, and *they do not have to be decision problems*.

The precise definition here is that *a problem `X` is NP-hard, if there is an NP-complete problem `Y`, such that `Y` is reducible to `X` in polynomial time*.

But since any NP-complete problem can be reduced to any other NP-complete problem in polynomial time, all NP-complete problems can be reduced to any NP-hard problem in polynomial time. Then, if there is a solution to one NP-hard problem in polynomial time, there is a solution to all NP problems in polynomial time.

Example

The *halting problem* is an NP-hard problem. This is the problem that given a program `P` and input `I`, will it halt? This is a decision problem but it is not in NP. It is clear that any NP-complete problem can be reduced to this one. As another example, any NP-complete problem is NP-hard.

My favorite NP-complete problem is the Minesweeper problem.

P = NP

This one is the most famous problem in computer science, and one of the most important outstanding questions in the mathematical sciences. In fact, the Clay Institute is offering one million dollars for a solution to the problem (Stephen Cook's writeup on the Clay website is quite good).

It's clear that P is a subset of NP. The open question is whether or not NP problems have deterministic polynomial time solutions. It is largely believed that they do not. Here is an outstanding recent article on the latest (and the importance) of the P = NP problem: The Status of the P versus NP problem.

The best book on the subject is Computers and Intractability by Garey and Johnson.

edited May 24 '18 at 9:01

ice1000
3,267 ● 4 ● 16 ● 46

answered Dec 7 '09 at 1:46

jason
194k ● 27 ● 359 ● 483

31 @Paul Fisher: I'll show that SAT is reducible to the halting problem in polynomial time. Consider the following algorithm: given as input a proposition `I` over `n` variables, try all `2^n` possible assignments to the variables and halt if one satisfies the proposition and otherwise enter an infinite loop. We see that this algorithm halts if and only if `I` is satisfiable. Thus, if we had a polynomial time algorithm for solving the halting problem then we could solve SAT in polynomial time. Therefore, the halting problem is NP-hard. – jason Dec 7 '09 at 3:52

5 @Jason - You can't reduce a decidable problem to an undecidable problem in that manner. Decidable problems have to result in a definitive yes or no answer in order to be considered to be decidable. The Halting Problem does not have a definitive yes or now answer since an arbitrary answer might throw any solution into a loop. – rjzii Dec 13 '09 at 2:48

11 @Rob: Yes, I can. The definition of reducible does not require that the problem being reduced to be solvable. This is true for either many-one reductions or Turing reductions. – jason Dec 13 '09 at 13:12

5 @Rob: Well, okay, if you want to continue this. First, "Decidable" is not synonomous with "decision problem" as you've used it. "Decidable" means, roughly, that there is an "effective method" for determining the answer. "Effective method", of course, has a technical definition. Moreover, "decidable" can also be defined in terms of "computable functions." So, the halting problem is a decision problem ("Does this program halt?" is a yes/no question) but it is undecidable; there is no effective method for determining whether or not an instance of the halting problem will halt. – jason Dec 13 '09 at 23:16

20 Using Halting problem as a "classic example" of NP-hard problem is incorrect. This is like saying: "Pacific Ocean is a classic example of a salt water aquarium." – Michael Mar 11 '14 at 21:18