P, NP, etc.: Brief Summary

Some algorithms are feasible, some are not. This we studied in CS2:

- an algorithm that requires linear time or quadratic time is usually practically feasible;
- on the other hand, an algorithm that requires time 2^n for inputs of size n is not practically feasible: for reasonable inputs of size $n = 300$ it would require more computation time than the lifetime of the Universe.

What is feasible: a common sense description. We say that an algorithm is feasible if for all inputs of reasonable length, it requires reasonable time.

This definition is not precise. Indeed, what is "reasonable" is not defined in precise terms.

A formal (precise) definition. An algorithm A is called *feasible* if it is polynomial-time, i.e., if there exists a polynomial $P(n)$ such that for every input x of length $\text{len}(x)$, its running time $t_A(x)$ does not exceed $P(\text{len}(x))$:

$$t_A(x) \le P(\text{len}(x)).$$

This definition is not perfect:

- if an algorithm requires time $10^{40} \cdot n$ for inputs of length n, then it is a linear-time algorithm and thus, feasible in the sense of the formal definition, but clearly not practically feasible: even for $n = 1$ it requires 10^{40} steps, not a practical amount;
- on the other hand, if an algorithm requires time $\exp(10^{-18} \cdot n)$, then formally it is exponential, so it grows faster than any polynomial and is, thus, not formally feasible; on the other hand, for even Exabyte-size inputs, with $n = 10^{18}$ that includes all the world's knowledge, this algorithm requires $\exp(1) = 2.7\ldots$ steps – a quite doable amount.

This definition is not perfect, but it is the best we can. If one of you can propose a better formal definition, that will be great!

Class P. The class of all the problems that can be solved by feasible (= polynomial-time) algorithms is usually denoted by P.

Class NP. In many practical situations – e.g., when solving a system of equations:

- we do not know how to solve the problem, but
- once someone presents us with a candidate solution, we can feasibly check whether this is indeed a solution;

e.g., for equations, substitute the solution into the left-hand side and into the right-hand side and check that the resulting values are indeed equal.

The class of all the problems for which:

- once we have a candidate for a solution,
- we can check, in feasible time, whether this candidate is a solution,

is called NP.

The abbreviation NP comes from *non-deterministic polynomial*. Indeed, as we remember, a word is accepted by a non-deterministic automaton if, once we guess how to proceed, we can get to the final state. In other words, non-deterministic means that to succeed, in addition to computations, we can make a guess.

Similarly have, we can solve an NP problem in polynomial (feasible) time if we guess a solution; after that, all that remains is to check that this guess is correct.

Is P equal to NP? Can every problem from the class NP be solved in polynomial time? In other words, is P – the class of all problems that can be solved in polynomial time – equal to NP?

No one knows, it is a known open problem. Most computer scientists believe that P is different from NP.

What *do* we know?

- We *do not* know whether P is equal to NP, but
- we *do* know that there are some problems in the class NP which are more difficult than others – in the sense that every other problem from the class NP can be feasibly reduced to this problem.

Such problems are known as *NP-complete*.

What does "reduction" mean: first example. Suppose that you have a method that, given three real numbers a, b, and c, finds all x for which

$$a \cdot x^2 + b \cdot x + c = 0.$$

What if now:

- we know the two values p and q, and
- we want to find all the values x for which $p \cdot x + q = 0$.

We can, of course, write a new software, but it much easier to use the existing method, and just plug in $a = 0$, $b = p$, and $c = q$.

In other words, we *reduce* the problem of solving linear equations to the problem of solving quadratic equations.

What does "reduction" mean: second example. What if we want to solve an equation $a \cdot y^4 + b \cdot y^2 + c = 0$?

If we introduce a new variable $z = y^2$, then for this new variable, we get the equation $a \cdot z^2 + b \cdot z + c = 0$. We can then:

- use the quadratic-equation-solving-method to find z, and
- compute $y = \sqrt{z}$.

In this case, we also reduce the problem of solving the equations $a \cdot y^4 + b \cdot y^2 + c = 0$ to the problem of solving quadratic equations.

NP-hard. Sometimes, there is a problem:

- which is not necessarily itself in the class NP, but
- to which every problem from the class NP can be reduced.

Such problems are called *NP-hard.*

An example of an NP-complete problem: propositional satisfiability:

- *given:* a propositional formula, i.e., any expression obtained from Boolean variables by using "and" (&& in Java), "or" (|| in Java), and "not" (! in Java) – e.g., $!(a \,||\, !b) \;\&\&\; (!a \,||\, b)$;
- *find:* the values of the Boolean variables that make the given formula true.

Why is this problem interesting? Because when we test a program with branching, we need to make sure that we have tested both branches. For this purpose, we need to find the values of the variables for which the corresponding condition is true. This is exactly what propositional satisfiability is about.

What do we gain and what do we lose when we prove that a problem is NP-complete? A positive consequence is that if we have a good algorithm for solving some cases of the problem, then we automatically get good algorithms for all other problems from the class NP – and many good algorithms have been obtained this way. A negative consequence is that, unless it turns out that P = NP, we cannot have a feasible algorithm for solving all particular cases of this problem.

An even more brief summary. Let us summarize what we have learned:

- P is the class of all the problems that can be solved in polynomial (= feasible) time.

- NP is the class of all the problems for which, once you have a candidate for a solution, you can check, in polynomial time, whether this candidate is indeed a solution.
- A problem from the class NP is called NP-complete if every problem from the class NP can be reduced to this problem.
- A problem is called NP-hard if every problem from the class NP can be reduced to this problem. *Comment:* the difference from NP-completeness is that an NP-hard problem may not be from the class NP.
- Example of an NP-complete problem – propositional satisfiability:
  - *given:* a propositional formula, i.e., any expression obtained from Boolean variables by using "and", "or", and "not",
  - *find:* the values of the Boolean variables that make the given formula true.
- Is P equal to NP? It is an unsolved open problem. Most computer scientists believe that P is different from NP.