13.1

P and NP

So far in the course we studied algorithms that solve computational problems efficiently. We saw linear time alg ($O(n)$), quasi-linear ($O(n \log n)$), quadratic $O(n^2)$ and cubic ($O(n^3)$). Common to all these alg. is that they take polynomial time to run as function of the input size.

Complexity theory tries to understand and classify computational problems. ~~for this~~ In particular we wish to understand which problems can be solved efficiently. One can define "efficient" in many ways. E.g. as being computable in linear time. But clearly, some problems take more than linear time. (although we can only prove this in ~~ex~~ very special cases). A more natural definition is that of being solvable in polynomial time. Clearly an alg. that takes time n^{1000} is not very efficient and could not be run in practice, but the definition still makes sense as

1. in many cases we do get better algorithms and
2. it is closed under composition. For example, if a poly-time alg. runs another poly-time alg. as a subroutine then the result is still ~~po~~ a poly-time alg.
3. Different architectures can solve basic problems more efficiently and get speedup, but when we study the asymptotic complexity as the input size grows, it is still polytime on ~~both~~ different machines or on neither.

The class of all computational problems that have poly-time alg. is denoted P. Notice that when we speak of a comp. prob. we mean to say that it is a problem that makes sense for growing input sizes. E.g. sorting ~~can be thought of as a collection~~ makes sense for any input size. In matrix-multiplication the input size is $2n^2$ (for multiplying two $n \times n$ matrices and again we can study what happens when $n \to \infty$.

However many important problems ~~that~~ are not known to be in P nor are believed to be there.
For example consider the vertex cover problem:
Input: A graph (undirected) and a number k.
Goal: determine if $\exists\, v_1, \dots, v_k$ s.t $\forall e \in E$ one of its end point is in $\{v_1 \dots v_k\}$

In other words, we wish to find a minimal set of vertices that touch all edges.
P (and NP) concern decision problem, i.e., YES/NO problems, this is why we don't ask for searching the minimum size, but rather ask whether it's $\le k$.
(with binary search we can find best size...).

This is important as it abstracts many scenarios. ~~Eg~~
We don't know of an efficient alg. for solving this problem. On the other hand, notice that if one would claim that such a cover exists and would present us with a solution then we'll be able to verify it efficiently.

Another example is independent-set (IS).
An ind. set. in a graph G is a subset $I \subseteq V$ s.t there are no edges between vertices in I.
Input: G, k
Goal: determine whether G has IS of size $\ge k$.

Again, no efficient alg. known, but a solution can be verified efficiently.

12.3

Another important problem is 3SAT.
Here we study boolean formulas of the form $C_1 \wedge C_2 \wedge \ldots \wedge C_m$
where each C_i is a clause on 3 variables $x \vee y \vee \bar{z}$.
Recall: each var can take a T/F value.
$x \vee y \vee \bar{z}$ ~~is~~ evaluates to T if $x = T$ or $y = T$ or $\bar{z} = T$ ($\Leftrightarrow z = F$).
Example: $(x \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee \bar{y} \vee \bar{z}) \wedge (y \vee \bar{z} \vee \bar{x})$
$x = T$, $y = F$, $z = T$ gives a true value. Thus, it is a satisfying assignment.

Input: a 3SAT formula on n vars
Goal: determine whether $\exists$ a sat. assignment.

No efficient alg. known, but a solution can be verified easily.

So we saw three diff. problems that have no efficient alg. but s.t. a solution for them can be verified efficiently. Soduko (say $n \times n$ size over $S = [n]$) is another such problem.

NP is the class of decision problems for which a solution can be verified efficiently.
NP stands for nondeterministic poly-time, where the nondeterminism comes from the need to "guess" a solution or from being handed one "nondeterministically".

The amazing fact about the problems that we mentioned is that if one of them is in P then all of NP is!
The famous P vs. NP problem asks exactly this: is P = NP?

Such problem, whose containment in P implies P=NP, are called NP-hard problems. An NP-hard problem that is also in NP is called an NP-complete problem.

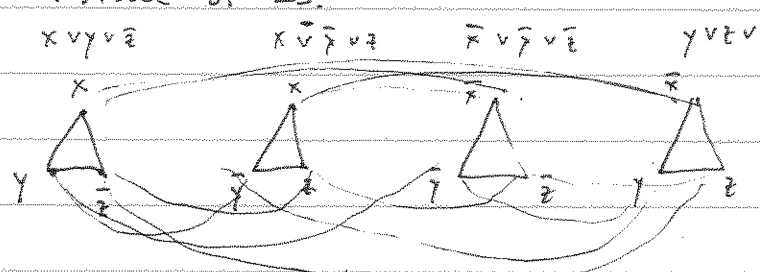
<u>Thm:</u> 3SAT, IS, Vertex Cover are NP complete.

We will not prove this, ~~but~~ as it requires more formal definitions than what we have given (and more time) but rather explain how one proves such a thing. The important concept is that of a REDUCTION between computational problem. That is, it is a poly-time computable function that takes inputs to one problem and transforms them to inputs to a second problem.

We already seen a reduction between SSSP and SDSP. Given an input to SDSP we reverse the edge directions (can be done easily) and solve the SSSP problem on the resulting graph.

Here is another reduction: $S \subseteq V$ is a vertex cover iff $V \setminus S$ is ind. set. Thus, if we can solve $IS(G,k)$ then we can solve $VC(G, n-k)$. and vice versa.

Showing that $3SAT \leq IS$ is more tricky. We will show given a formula $\bigwedge_{i=1}^{m} C_i$ how to get an instance of IS.



~~$x \vee y \vee \bar{z}$~~

Each clause gives a triangle. In addition each appearance of v is connected to $\bar{v}$

13.5

~~Claim~~ We get a graph on $3m$ vertices.

Claim: G has IS of size m iff $\bigwedge_{i=1}^{m} C_i$ is sat.

Pf: A sat. assgn. picks (at least) one var from each clause.
we never pick both v and $\bar{v}$. (sat ass. $\Rightarrow$ IS)
(IS $\Rightarrow$ SAT). An IS must contain exactly one var from each $\triangle$
This gives an ass. in the natural way (no ambiguity because of $v - \bar{v}$ edges).

The general pf. that 3SAT is NP complete follows from showing that the steps of any poly-time alg. can be encoded using a boolean formula.

What about problems between P and NP:

Input: An n digit number N, integer k.
Goal: Determine whether $N = P \cdot Q$ ~~for two prime numbers P, Q~~ where $P < k$.

This is in NP since there is a (nontrivial!) alg. for deciding primality.
(we can give N's ~~p~~ prime factors and multiplicities as proof).

It is not believed to be NP complete as it will imply other things that are not believed (but if P=NP then it is complete...).

Yet, it is believed to be hard and the famous RSA encryption scheme is based on it.

13.6

Another fun fact. Security of crypto currency, bitcoin, ethereum etc. is based on $P \neq NP$!

Lesson: when you can't find an efficient alg. try to find proof of NP completeness.

In practice: many NP problems are important and necessary so there are heuristics (alg. with no proof, or that work for some inputs) that are used in practice.

There is a famous SAT-solving competition between SAT-solvers. They work well in practice, but we know how to come up with hard inputs for them.