4/27 NP-completeness and Reductions:

Polynomial Reductions: consider two decision problems A and B. We say A is polynomially reducible to B if there exists a polynomial-time algorithm that converts each input a_i to A to another input b to B such that the answer to input a for problem A is "yes" if and only if the answer to input b for problem B is "yes". (here we assume the size of b is also polynomial in the size of a).

Some simple theorems: For example: the reductions from undirected graphs to directed ones for shortest p

thm1: If A is polynomially reducible to B, and there is a polynomial-time algorithm for B, then there is a polynomial-time algorithm to A as well.

Pf: first reduce the input of A to B and then solve it.

Thm2: If A is polynomially reducible to B and B is polynomially reducible to C, then A is polynomially reducible to C. (Transitivity)

Pf: Just compose the two polynomial-time algorithms in the reductions.

Now we define two classes, which not only contain numerous important problems (all equivalent to one another) that are not known to be in P, but also contain the hardest problems in NP.

Definition: A problem X is called NP-hard problem if every problem is NP is polynomially reducible to X.

Definition: A problem X is called NP-complete if 1) X belongs to NP
2) X is NP-hard.

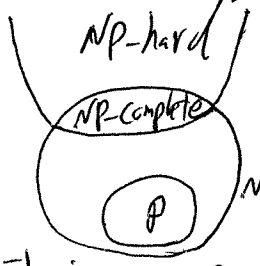
Note that definition of NP-hardness implies that if any NP-hard problem is ever proved to belong to P, then that proof would imply that NP=P. (by thm 1)
[and You win the Million PRIZE]

Cook [1971] proved that there is at least one NP-complete problem Satifiability (SAT,

Now our job is much easier. why?

Thm 3: Problem X is NP-complete if (1) X belongs to NP (2) For some problem Y that is NP-complete, Y is polynomially reducible to X (we can transform each input of Y to an input of X)

Pf: by condition 2 of the definition of NP-Completeness, every problem in NP is polynomially reducible to Y. But since Y is polynomially reducible to X and reducibility is transitive (by Thm 2), every problem in NP is polynomially reducible to X as well.



Satisfiability (SAT) problem:

let S be a boolean expression in Conjunctive Normal Form (CNF).

that is S is a And (product) of several ORs (sum), e.g. $S=(x+y+\bar{z})(\bar{x}+y+z)(\bar{x}+\bar{y}+\bar{z})$ if $P \neq NP$

Each variable is either 0 (false) or 1 (true).

One can show any boolean expression can be transformed into CNF.

A boolean expression is said to be <u>satisfiable</u> if there exists an assignment of 0s and 1s to its variables such that the value of the expression is 1. The <u>SAT</u> problem is to determine whether a given expression is satisfiable (without necessarily finding a satisfying assignment). For example for S above $x=1, y=1$ and $z=0$ satisfy it.

We call an assignment of 0s and 1s to the variables of a Boolean expression a <u>truth assignment</u>.

Note that using a back tracking algorithm is time $O(2^n n)$ we can solve SAT

<u>Cook's Theorem</u>: The SAT is NP-complete. (note that SAT is NP is trivial).

The proof is complicated and uses modeling of a non-deterministic computer with SAT.

Note that a few other problems such as graph coloring and Bin-packing that you have seen so far are NP-complete as well (that's the reason that we designed approximation algorithm for them).

Now let's see more examples.

3SAT is NP-Complete

3SAT problem: Given a Boolean expression in CNF such that each clause contains exactly three variables, determine whether it is satisfiable.

The problem seems easier that SAT since there is the additional requirement of three variables per clause.

Pf: First the problem is clearly in NP since given a certificate of a truth assignment we can verify it easily in polynomial time. Let $C=(x_1+x_2+\cdots+x_k)$ be an arbitrary clause of the input to SAT with k variables. If $k=3$ we are done. We write each variable in its "positive" form (i.e. we do not use $\overline{x_i}$) only for convenience of the problem. Now we show how to replace any clause C of SAT with several clauses of 3-SAT each with exactly three variables. If $k=2$, $C=x_1+x_2$ then we replace it with $C'=(x_1+x_2+z)(x_1+x_2+\overline{z})$. If $k=1$, $C=x_1$ then $C'=(x_1+y+z)(x_1+\overline{y}+z)(x_1+y+\overline{z})\cdot(x_1+\overline{y}+\overline{z})$ where both y and z are new variables. Note that C is satisfiable iff C' is satisfiable. Now if $k\geq 4$, then

$$C'=(x_1+x_2+y_1)(x_3+\overline{y}_1+y_2)(x_4+\overline{y}_2+y_3)\cdots(x_{k-1}+x_k+\overline{y}_{k-3}).$$

Now if C is satisfiable, then one of the x_is must be set to 1. Then we can set the values of the y_is in C' such that all clauses in C' is satifried as well. E.g. if $x_3=1$ then we set $y_1=1$ (which takes care of the first clause) $y_2=0$ (the second clause is ok since $x_3=1$, then we set the rest of y_is to zero. In general if $x_i=1$ then we set $y_1, y_2, \ldots y_{i-2}$ to be 1, and the rest to be 0, which satisfies C'. Conversely, if C' is satisfiable, then we claim that at least one of the x_is must be 1. otherwise if they are all zero, then the expression becomes $y_1(\overline{y}_1+y_2)(\overline{y}_2+y_3)\cdots(\overline{y}_{k-3})$ which is clearly unsatisfiable, since $y_i=1$ but then $\overline{y}_{k-3}\neq 0$ which is bad.