==
CSC 373 Lecture Summary for Week 9 Winter 2019
==

NP-completeness:

- Recall: decision problem D is "NP-complete" iff
  1. D in NP
  2. D is "NP-hard": for all D' in NP, D' -p> D.

- Property: If P != NP and D is NP-complete, then D not in P.

- Updated "picture of the world" with NP-hard extending beyond NP but intersecting with it (the intersection is "NP-complete").

Cook's Theorem: [properly Cook-Levin Theorem]

* Circuit-SAT: Given a circuit with a single output gate, is there some setting of the inputs that will make the output equal to 1?

* SAT: Given a propositional formula \phi (written using propositional connectives negation, and, or, implication), is there some setting of the variables that will make \phi true (in which case \phi is said to be "satisfiable")?

* CNF-SAT: Given a propositional formula \phi in Conjunctive Normal Form (also called product of sums), is \phi satisfiable?
  Note this means \phi has the form C_1 /\ C_2 /\ ... /\ C_k, where each "clause" C_i = a_1 \/ a_2 \/ ... \/ a_r, where each "literal" a_j is either a variable (x) or negated variable (~x). For example:
  ```
  (x1 \/ ~x2) /\ ~x3 /\ (~x1 \/ x2 \/ x3 \/ x2)
  ```

* 3SAT: Given a propositional formula \phi in 3-CNF (CNF where each clause contains exactly 3 literals), is \phi satisfiable?

* Cook-Levin Theorem: SAT is NP-complete.

  - SAT in NP:
    Given \phi,c, where c is a setting of values (True/False) for the variables of \phi:
    Output the value of \phi under the setting given by c.
    This can be carried out in polynomial time: given a formula \phi and a setting of its variables, just substitute the values for each variable and then evaluate each connective one-by-one, from the inside out. Moreover, if \phi is satisfiable, then there is some value of c that will make this verifier output yes (when c = a setting that makes \phi true); and if \phi is not satisfiable, then this verifier will output "no" for every possible value of c (since no setting makes \phi true).

    The same reasoning shows that CNF-SAT and 3SAT also belong to NP.

  - SAT is NP-hard (main idea):
    Let D be any problem in NP. By definition, there is a polytime verifier V(x,c) for D. This polytime verifier can be implemented as a circuit with input gates representing the values of x and c. For any input x for D, we can hard-code the value of x into this circuit in such a way that there is a value of the certificate for which the verifier outputs "yes" iff there is some setting of the input gates corresponding to c that make the circuit output 1. It's possible to

show that this transformation can be carried out in polynomial time (as a function of the size of x), and it's also possible to show that this circuit can then be translated into a formula in CNF (in polytime) such that settings of the circuit's input gates correspond to settings of the formula's variables.

This shows directly that CNF-SAT and SAT are both NP-hard. It leaves open the question for 3SAT, for now...

NP-hardness:

- In general, how do we show decision problem D is NP-hard? Don't want to re-prove Cook's Theorem from scratch for each problem!

- Note: -p> is transitive -- if A -p> B and B -p> C, then A -p> C.

- To show D is NP-hard, it is sufficient to find some NP-hard problem D' and prove D' -p> D because D' NP-hard implies for all D'' in NP, D'' -p> D' so D'' -p> D (by transitivity of -p>).

NP-completeness examples

- SUBSET-SUM: Given a finite set of positive integers S and a positive integer target t, is there some subset S' of S whose sum is exactly t, i.e., -] S' (_ S, SUM_{x in S'} x = t?

- SS is NPc:
  SS in NP because it takes polytime to verify that the certificate represents a subset of S whose sum is t (addition of two numbers takes linear time; addition of k numbers takes time proportional to the sum of the bit-lengths of all the numbers).
  SS is NP-hard because 3SAT -p> SS:
  Given formula \phi = (a1 \/ b1 \/ c1) /\ ... /\ (ar \/ br \/ cr) where ai,bi,ci in {x1,~x1,...,xs,~xs}, construct numbers as follows:
    . For j = 1,...,s,
      number y_{2j-1} = 1 followed by s-j 0s followed by r digits where k-th next digit equals 1 if xj appears in clause C_k, 0 otherwise (corresponds to literal x_j);
      number y_{2j} = 1 followed by s-j 0s followed by r digits where k-th next digit equals 1 if ~xj appears in clause C_k, 0 otherwise (corresponds to literal ~x_j).
    . For j = 1,...,r,
      number y_{2s+2j-1} = 1 followed by r-j 0s and
      number y_{2s+2j} = 2 followed by r-j 0s
      (both correspond to clause C_j).
    . Target t = s 1s followed by r 4s.
  Clearly, this can be constructed in polytime.

  Example of reduction for
  \phi = (x1 \/ ~x2 \/ ~x4) /\ (x2 \/ ~x3 \/ x1) /\ (~x3 \/ x4 \/ ~x2):

```
S = { y_01 = 1000110,  [corresponds to  x1]
      y_02 = 1000000,  [corresponds to ~x1]
      y_03 =  100010,  [corresponds to  x2]
      y_04 =  100101,  [corresponds to ~x2]
      y_05 =   10000,  [corresponds to  x3]
      y_06 =   10011,  [corresponds to ~x3]
      y_07 =    1001,  [corresponds to  x4]
      y_08 =    1100,  [corresponds to ~x4]
      y_09 =     200,  [corresponds to  C1]
      y_10 =     100,  [corresponds to  C1]
```

```
    y_11 =      20,  [corresponds to  C2]
    y_12 =      10,  [corresponds to  C2]
    y_13 =       2,  [corresponds to  C3]
    y_14 =       1}  [corresponds to  C3]
       t = 1111444
```

If \phi is satisfiable, then there is a setting of variables such that each clause of \phi contains at least one true literal. Consider the subset S' = {numbers that correspond to true literals}. By construction, SUM_{x in S'} x = s 1s followed by r digits, each one of which is either 1, 2, or 3 (because each clause contains at least one true literal). This means it is possible to add suitable numbers from {C1,D1,...,Cr,Dr} so that the last r digits of the sum are equal to 4, i.e., there is a subset S' such that SUM_{x in S'} x = t.

If there is a subset S' of S such that SUM_{x in S'} x = t, then S' must contain exactly one of {xj,~xj} for j = 1,...,n, because that is the only way for the numbers in S' to add to the target (with a 1 in the first s digits). Then, \phi is satisfied by setting each variable according to the numbers in S': for each clause j, the corresponding digit in the target is equal to 4 but the numbers Cj and Dj together only add up to 3 in that digit; this means that the selection of numbers in S' must include some literal with a 1 in that digit, i.e., clause j contains at least one true literal.

- VERTEX-COVER:
  Input: Undirected graph G = (V,E), positive integer k.
  Output: Does G contain a vertex cover of size k, i.e., a subset C of k vertices such that each edge of G has at least one endpoint in C?

- VERTEX-COVER (VC) is NPc:
  VC in NP: Given G,k,c, verify in polytime that c represents a vertex cover of size k in G.
  VC is NP-hard: 3SAT -p> VC.
  Given \phi = (a1 \/ b1 \/ c1) /\ ... /\ (ar \/ br \/ cr), where ai,bi,ci in {x1,~x1,x2,~x2,...,xs,~xs}, construct G=(V,E) and k such that \phi satisfiable iff G contains vertex cover of size k, as follows:

```
    k = s + 2r
    V = { a1,b1,c1, ..., ar,br,cr, x1,~x1, ..., xs,~xs }
    E = { (xi,~xi) : 1 <= i <= s } U
        { (ai,bi),(bi,ci),(ci,ai) : 1 <= i <= r } U
        { (l,x) : l = ai or bi or ci, and x = xj or ~xj
                       corresponding to l }
```

For example, if
\phi = (x1 \/ ~x2 \/ ~x4) /\ (x2 \/ ~x3 \/ x1) /\ (~x3 \/ x4 \/ ~x2),
then a1=x1, b1=~x2, c1=~x4, a2=x2, b2=~x3, c2=x1, a3=~x3, b3=x4, c3=~x2
so k = 4 + 2*3 = 10

```
V = {a1,b1,c1, a2,b2,c2, a3,b3,c3, x1,~x1, x2,~x2, x3,~x3, x4,~x4}
E = { (x1,~x1), (x2,~x2), (x3,~x3), (x4,~x4),
      (a1,b1), (b1,c1), (c1,a1), (a1,x1), (b1,~x2), (c1,~x4),
      (a2,b2), (b2,c2), (c2,a2), (a2,x2), (b2,~x3), (c2,x1),
      (a3,b3), (b3,c3), (c3,a3), (a3,~x3), (b3,x4), (c3,~x2) }
```

Clearly, construction can be done in polytime (with one scan of \phi).

Also, if \phi is satisfiable, then there is an assignment of truth values that make at least one literal in each clause true. Pick a cover C as follows: for each variable, C contains xi or ~xi, whichever is true under the truth assignment; for each clause, C contains every literal except one that's true (pick arbitrarily if more than one true literal).

C contains exactly s+2r vertices and is a cover: all edges (xi,~xi) are covered; all edges in clause triangles are covered (because we picked two vertices from each triangle); all edges between "clauses" and "variables" are covered (two from inside triangle, one from true literal for that clause).

Finally if G contains a cover C of size k=s+2r, C must contain at least one of xi or ~xi for each i (because of edges (xi,~xi)) and at least two of ai,bi,ci for each i (because of triangle), so only way for C to have size s+2r is to contain exactly one of xi or ~xi and exactly two of ai,bi,ci, for each i. Since C covers all edges with only two vertices per triangle, the third vertex in each triangle must have its "outside" edge covered because of xi or ~xi. If we set literals according to choices of xi or ~xi in C, this will make formula \phi true: at least one literal will be true in each clause (because at least one edge from "variables" to "clauses" is covered by the variable in C).

The material below was not covered during lectures -- it is provided here for your reference.

Extra example: 3SAT is NPc.
3SAT in NP because it’s a special case of SAT.
CNF-SAT -p> 3SAT:
Given \phi (a CNF formula), construct \phi’ (a 3-CNF formula) such that \phi is satisfiable iff \phi’ is satisfiable, as follows. Note that it is not necessary to make \phi and \phi’ logically equivalent in order to achieve this. For each clause C of \phi:

- If C = (a1), then replace C with (a1 \/ a1 \/ a1).

- If C = (a1 \/ a2), then replace C with (a1 \/ a1 \/ a2).

- If C = (a1 \/ a2 \/ a3), then leave C the same.

- If C = (a1 \/ a2 \/ ... \/ ar) where r > 3, then replace C with
  (a1 \/ a2 \/ z1) /\ (~z1 \/ a3 \/ z2) /\ (~z2 \/ a4 \/ z3) /\ ...
  /\ (~z{r-4} \/ a{r-2} \/ z{r-3}) /\ (~z{r-3} \/ a{r-1} \/ ar),
  where z1, z2, ..., z{r-3} are new variables (not in \phi).

For example, if
    \phi = (x1 \/ x2) /\ (~x1) /\ (x2 \/ ~x3 \/ x3 \/ x5 \/ ~x4)
then
    \phi’ = (x1 \/ x1 \/ x2) /\ (~x1 \/ ~x1 \/ ~x1) /\
            (x2 \/ ~x3 \/ z3) /\ (~z3 \/ x3 \/ z4) /\ (~z4 \/ x5 \/ ~x4)

Clearly, this transformation can be carried out in polytime: at most, each clause of length r gets replaced with O(r) 3-clauses using O(r) new variables.

Also, if \phi is satisfiable, then there is an assignment of truth values to the variables of \phi that makes at least one literal true in each clause of \phi. This assignment can be extended to include values for the new variables of \phi’ that will make each clause of \phi’ true:

- For 1-/2-/3-clauses of \phi with at least one true literal, the corresponding clause in \phi’ is also true because it contains the same literals, at least one of which is true.

- For r-clauses of \phi with at least one true literal, say the original clause is (a1 \/ a2 \/ ... \/ ar) and the true literal is ai. Then pick values for the new variables as follows:

. if i=1 or i=2, then (a1 \/ a2 \/ z1) is satisfied so pick z1 = z2 = ... = z{r-3} = false to satisfy every other clause;

. if i=r-1 or i=r, then (~z{r-3} \/ a{r-1} \/ ar) is satisfied so pick z1 = z2 = ... = z{r-3} = true to satisfy every other clause;

. if 2 < i < r-1, then (~z{i-2} \/ ai \/ z{i-1}) is satisfied so pick z1 = z2 = ... = z{i-2} = true to satisfy the first i-2 clauses and pick z{i-1} = zi = ... = z{r-3} = false to satisfy the last r-i-1 clauses.

For example, if x3 = true satisfies (x2 \/ ~x3 \/ x3 \/ x5 \/ ~x4), then pick z1 = true and z2 = false to satisfy

(x2 \/ ~x3 \/ z1) /\ (~z1 \/ x3 \/ z2) /\ (~z2 \/ x5 \/ ~x4)

(the first clause is satisfied by z1 = true, the second clause is satisfied by x3 = true, the last clause is satisfied by z2 = false).

Finally, if \phi' is satisfiable, then the assignment of values to the variables of \phi' must include values to the variables of \phi that satisfy \phi:

- If the new clauses
  (a1 \/ a2 \/ z1) /\ (~z1 \/ a3 \/ z2) /\ (~z2 \/ a4 \/ z3) /\ ...
  /\ (~z{r-4} \/ a{r-2} \/ z{r-3}) /\ (~z{r-3} \/ a{r-1} \/ ar)
  are satisfied, then let zi be the first new variable set to false (so either i = 1 or z1 = z2 = ... = z{i-1} = true):
  . if i = 1, then (a1 \/ a2 \/ z1) can only be satisfied by setting a1 = true or a2 = true;
  . if i > 1, then (~z{i-1} \/ a{i+1} \/ zi) can only be satisfied by setting a{i+1} = true;
  in all cases, one of the original literals must be set to true so the original clause (a1 \/ a2 \/ ... \/ ar) is also satisfied.

- If the new clause (a1 \/ a2 \/ a3) is satisfied, then the original clause is also satisfied because it's the same, and similarly for the new clauses (a1 \/ a1 \/ a2) and (a1 \/ a1 \/ a1), because they are logically equivalent to the original clauses.

We have shown that any CNF formula \phi can be transformed in polytime to a 3-CNF formula \phi' such that \phi is satisfiable iff \phi' is satisfiable; this completes the polytime reduction from CNF-SAT to 3SAT.

Note: Careful with directions! Trivially, 3SAT -p> CNF-SAT (3SAT is special case of CNF-SAT). But in this case, we need other direction, transforming instances of general problem into instances of restricted problem.

For Next Week

* Readings: No readings in textbook for next week.
* Self-Test: Think about exercises 8.1 and 8.2.