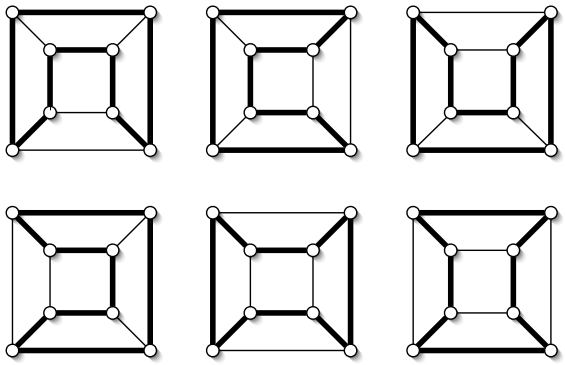
Homework #4 Solutions

1. Let G be the undirected graph with 8 vertices and 12 edges formed by the edges of a cube. How many undirected Hamiltonian cycles does G have?

   Answer. Six:



   □

2. Show that Primality is in P if we give the input in unary rather than binary. In other words, show that $L = \{1^p : p \text{ is prime}\}$ is in P.

   Now consider Primality when the input is given in binary. Is it obviously in NP? Is its complement, Compositeness (i.e., the property of not being a prime)? In fact, both of these are in NP, and it was proved two years ago that they are in P; but I want you to tell me which one is *obviously* in NP.

   Answer. If the input is given in unary, the size of the input is p. We can easily check that p is prime in time polynomial in p; even the most simple-minded algorithm, such as dividing by all numbers less than p, does this. If the input p is given in binary, on the other hand, the size of the input is $n = \log_2 p$. The question now becomes whether we can tell whether p is prime in time polynomial in $\log_2 p$, which is much harder.

   However, Compositeness is clearly in NP, since to prove to you that p is *not* prime, all I have to do is give you two numbers $x, y > 1$ such that $xy = p$; you can easily multiply these n-digit numbers in time polynomial in n and confirm that their product is p. Thus the pair (x, y) is a certificate for p's compositeness. (Even a single factor of p would be a certificate, since you can divide p by it.) □

3. Two graphs G and H are *isomorphic* if their structure is the same, except that their vertices are permuted: formally, if there is a 1-1 and onto mapping ϕ from the vertices of G to the vertices of H such that u and v are connected in G if and only if $\phi(u)$ and $\phi(v)$ are connected in H. Prove that the property Graph Isomorphism (that is, the language $\{\langle G, H\rangle : G \text{ and } H \text{ are isomorphic graphs}\}$) is in NP. What do you think about Graph Non-Isomorphism? Is it in NP?

   Answer. A certificate for Graph Isomorphism is simply the mapping ϕ, i.e., a list giving $\phi(u)$ for each vertex u of G. We can clearly check in polynomial time (in fact, time linear in the number of vertices and edges) that 1) ϕ is 1-1 and onto, and 2) it preserves the edges as described above.

   Providing a certificate for Graph Non-Isomorphism isn't so easy. We can ask whether G and H have the same number of vertices of each degree, the same number of triangles, etc. But if they pass all these tests, we somehow have to prove that none of the $n!$ possible mappings work. Thus Graph Non-Isomorphism might not be in NP (while it is in co-NP by definition). □

4. Recall that the star operation on a language gives $L^* = \cup_{i \geq 0} L^i$ where L^i is the concatenation of L with itself i times. Show that P is closed under this operation: that is, if $L \in$ P, then $L^* \in$ P. The tricky part is that if I give you an input in L^*, you don't know where the boundaries between words in L are. Hint: build a table of which substrings of the input are in L, and then use dynamic programming.

   Answer. A substring s of w is in L^* if either $s \in L$ or $s = tu$ for two smaller substrings $t, u \in L^*$. For a bottom-up approach, start by checking the substrings of length 1, and build larger substrings until you either get stuck or succeed in building all of w. For a top-down approach, define a recursive function which checks whether $s \in L$ or s can be broken into smaller substrings in L^*, and memoize this function so that you don't recalculate it for substrings you've tried before. Both of these approaches work in polynomial time, since there are only $O(|w|^2)$ possible substrings to check. □

5. Show that if $L \in$ NP, then $L^* \in$ NP. Hint: this is easier than the previous problem.

   Answer. NP has the power to "guess" where the boundaries between words in L are. To put this more formally, we can provide a certificate for the statement $w \in L^*$ as follows: give a set of $k-1$ integers $1 \leq j_1 < j_2 < \cdots < j_{k-1} \leq |w|$, and a list of certificates (which exist since $L \in$ NP) for the k statements

$$\begin{aligned} w_0 \cdots w_{j_1} &\in L \\ w_{j_1+1} \cdots w_{j_2} &\in L \\ &\vdots \\ w_{j_{k-1}+1} \cdots w_{|w|} &\in L \end{aligned}$$

   □

6. Consider two problems, Short Path and Long Path. Both take input (G, u, v, k) where G is a graph, u and v are vertices, and k is an integer. Short Path asks whether there is a path in G from u to v of length *at most* k, and Long Path asks whether there is a path of length *at least* k. In both cases paths are not allowed to visit the same vertex twice. Show that Short Path is in P, but Long Path is NP-complete.

   Answer. Short Path is in P because we can find the shortest path using, say, Dijkstra's algorithm, and checking whether it is shorter than k (note that the shortest path never visits a vertex twice!) Long Path is in NP since the path is the certificate (we can easily check in polynomial time that it is a path, and that its length is k or more), and NP-complete since Hamiltonian Path (the variant where we specify a start and end node) is a special case of Long Path, namely where k equals the number of vertices of G minus 1. □

7. Recall that Not-All-Equal Satisfiability (NAE-SAT) is like SAT, but where we demand that every clause contains at least one true literal and at least one false one. NAE-3-SAT is the special case where each clause has exactly 3 literals. Show that NAE-3-SAT is NP-complete by reducing 3-SAT to it. Make sure you prove that your reduction works: that is, that a satisfying assignment for the original 3-SAT formula exists if, and only if, a satisfying assignment exists for the resulting NAE-3-SAT formula.

   Hint 1: it might be easier to first reduce 3-SAT to NAE-k-SAT for some $k > 3$ and then show how to break up NAE-k-SAT clauses into NAE-3-SAT clauses.

   Hint 2: unlike for SAT, for NAESAT the complement of a satisfying assignment is also satisfying, and this makes it initially confusing how to reduce SAT to NAESAT. Try "breaking the symmetry" by using one or more additional variables as references to define which value will correspond to True or False in the reduction. You may not use constants like T or F.

   Answer. NAE-3-SAT, like any variant of SAT, is in NP since the truth assignment is the certificate; we can check every clause in polynomial time to see if it is satisfied.

   Let's define a single "reference variable" z for the entire NAESAT formula. Then we represent each variable x_i in the 3-SAT formula with a variable y_i in the NAESAT formula, where x_i is true or false if $y_i \neq z$ or $y_i = z$ respectively. Then we can represent 3-SAT clause $(x_i \vee x_j \vee x_k)$ with a NAE-4-SAT clause

$$(y_i, y_j, y_k, z)$$

   since this clause is satisfied if and only if at least one of y's is different from z, i.e., if at least one of the x's is true. We can similarly represent $\overline{x}_i$ by $\overline{y}_i$.

   This shows that NAE-4-SAT is NP-complete. To reduce this to NAE-3-SAT, we use variables w to break each 4-variable clause into two 3-variable ones, just as for 3-SAT:

$$(y_i, y_j, y_k, z) = (y_i, y_j, w) \wedge (\overline{w}, y_k, z)$$

It is easy to check that this works, i.e., by setting w appropriately we can satisfy these two NAE-3-SAT clauses unless $y_i = y_j = y_k = z$. □

Alternate solution. Represent each x_i with a pair of variables y_i, z_i, where x_i is true if $y_i \neq z_i$; that is, $x_i = y_i \oplus z_i$. Then the 3-SAT clause $(x_i \vee x_j \vee x_k)$ becomes a set of four NAE-6-SAT clauses,

$$\begin{array}{c}(y_i, z_i, y_j, z_j, y_k, z_k)\\ (y_i, z_i, y_j, z_j, \overline{y}_k, \overline{z}_k)\\ (y_i, z_i, \overline{y}_j, \overline{z}_j, y_k, z_k)\\ (y_i, z_i, \overline{y}_j, \overline{z}_j, \overline{y}_k, \overline{z}_k)\end{array}$$

and we break these down into NAE-3-SAT clauses as above. If x_i, say, is negated, we flip z_i in each of these clauses. □

8. Show that Monotone NAE-3-SAT, where no variables are negated, is NP-complete. Hint: use two variables in the new formula for each variable of the old. You may repeat variables in a clause.

   Answer. For membership in NP, as for 3-SAT or NAE-3-SAT the truth assignment is the certificate, and we can check all the clauses in polynomial time.

   To reduce NAE-3-SAT to Monotone NAE-3-SAT, represent x_i by y_i and $\overline{x}_i$ by z_i. For this encoding to make sense, we have to make sure that y_i and z_i take opposite values; we can do this by including the clause (y_i, z_i, z_i) for each i, or, if we are not allowed to repeat variables in a clause, use new variables t_i, u_i, v_i and include

   $$(y_i, z_i, t_i), (y_i, z_i, u_i), (y_i, z_i, v_i), (t_i, u_i, v_i) \ .$$

   The last clause forces one of t_i, u_i, v_i to be false and one to be true; if y_i and z_i were both true or both false, then one of the first three clauses would be dissatisfied.

   We can then represent a NAE-3-SAT clause with some x's negated with a monotone clause in the y's and z's; for instance,

   $$(x_i, \overline{x}_j, x_k) = (y_i, z_j, y_k) \ .$$

9. Given a graph G with vertices V, a *cut* is a subset $S \subset V$. The size of the cut is the number of edges with one end in S and the other end in $\overline{S}$. MAX-CUT is the following problem: given a graph G and a number k, does G have a cut of size k or more? Show that MAX-CUT is NP-complete by reducing NAE-3-SAT to it.

   Hint: if the NAE-3-SAT formula has k clauses, represent each variable x with $3k$ vertices labelled x and another $3k$ vertices labelled $\overline{x}$, with $(3k)^2$ edges connecting all the x vertices to all the $\overline{x}$ vertices. Then represent each clause with a triangle connecting an appropriate triple of vertices. Figure out how large the cut corresponding

to a solution to the NAE-3-SAT formula is, and prove that a cut of this size exists if and only if the formula is satisfiable.

Answer. For membership in NP, the certificate is the set S. In polynomial time we check all the edges of G and count those "in the cut," i.e., those connecting vertices in S to vertices in $\overline{S}$, and see if there are k or more.

We represent setting x true (or false) by putting all $3k$ vertices corresponding to x in S and all those corresponding to $\overline{x}$ in $\overline{S}$ (or vice versa). This puts all $(3k)^2$ edges crossing from x to $\overline{x}$ in the cut. Then a satisfied clause triangle has 2 of its 3 edges in the cut, and an unsatisfied clause has none. Thus the size of a cut corresponding to a satisfying assignment is (where n is the number of variables)

$$M = (3k)^2 n + 2k \ .$$

If the formula is unsatisfiable, then respecting the "encoding" of truth values we can get a cut of size at most $(3k)^2 n + 2(k-1)$ since at least one clause is unsatisfied.

Now suppose we fail to respect the encoding. That is, for some x (or some $\overline{x}$) suppose that among its $3k$ vertices, there is some $v_1 \in S$ and some $v_2 \in \overline{S}$. Take whichever of these matches the majority of the vertices corresponding to $\overline{x}$ (or x); this gives at least $(3/2)k$ edges which are not in the cut. At most 2 edges of any triangle can be in the cut; even if we achieve this for all k triangles, we get a cut of size at most $(3k)^2 n + 2k - (3/2)k$. Thus in neither case do we achieve a cut of size M.

Note 1: Besides giving us an incentive to respect the encoding, these groups of $3k$ vertices allow us to avoid multiple edges between vertices: although it would be an absurd formula, we could have a single variable occur $3k$ times.

Note 2: as an alternate proof that MAX-CUT is NP-complete, there is an extremely simple reduction from Monotone NAE-3-SAT to it. Do you see it? □

10. Suppose I have a 3-SAT formula F. Define the *degree* of a variable x_i as the total number of times it appears in F (either positively or negatively). Show that the restricted version of 3-SAT in which every variable has degree at most 2 is in P. What degree do you think we need for NP-completeness?

    Answer. We will show that all such formulas are satisfiable, and therefore the following polynomial-time algorithm works: print "yes."

    First, we can assume without loss of generality that each variable appears twice, once positively and once negatively, in two different clauses. The reason for this is that if it appears once, or twice in the same way, we can give it that truth value and satisfy the clause(s) it appears in. If it appears both positively and negatively in the same clause, that clause is automatically satisfied.

    Now consider the following algorithm, which tries to satisfy the formula by setting one variable at a time. Whenever there is a *unit clause* — that is, a clause with a single

variable — satisfy it by setting its variable. Otherwise, set any variable to either value. This satisfies any clause in which the variable appears with this value, and shortens any clause in which it appears the other way. The only way a contradiction can arise is if there are two unit clauses which demand opposite things of the same variable, (x) and $(\overline{x})$. But since each variable occurs at most once each way, each step of this algorithm shortens at most one clause, and so there will never be more than one unit clause. Thus this algorithm will satisfy the entire formula without creating a contradiction.

Can you prove a similar result about degree 3? We hear from Prof. Moret that degree 4 is enough to make 3-SAT NP-complete. □

11. Point out and discuss the fallacy in the following "proof" that P $\neq$ NP: "To see if a 3-SAT formula is satisfiable, we need to look at 2^n possible truth assignments. This takes exponential time, so 3-SAT is not in P. But it is in NP, so P $\neq$ NP."

Answer. The fact that one algorithm takes exponential time doesn't prove that there is no other algorithm which is faster. Looking at all 2^n truth assignments might not be necessary; there might be some clever way of "zeroing in" on a small part of the solution space rather than searching all of it with brute force. □