1. Suppose an oracle has given you a magic computer, \( C \), that when given any Boolean formula \( B \) in CNF will tell you in one step whether \( B \) is satisfiable. Show how to use \( C \) to construct an actual assignment of satisfying Boolean values to the variables in any satisfiable formula \( B \). How many calls do you need to make to \( C \) in the worst case in order to do this?

For a formula \( B \) with \( n \) variables, we can construct an assignment of satisfying variables with \( n \) queries to \( C \). Assume our variables are labeled \( x_1, x_2, \ldots, x_n \).

We first ask if the formula is satisfiable when \( x_1 \) is true. If so, we set \( x_1 \) to be true and continue. If not, we set \( x_1 \) to be false and continue.

We repeat for every variable. In the last step, we will ask if our formula is satisfiable when \( x_n \) is true. If so, we have constructed a satisfying assignment. Otherwise, we evaluate our formula when \( x_n \) is false. If it evaluates to true, we have a satisfying assignment. If not, then no satisfying assignment is possible.

(Not required for this proof, but an interesting side note.) Note that this is the optimal solution, since there are \( 2^n \) possible truth assignments, and we get one bit of information with each query. So no matter how we arrange them, we need at least \( n \) queries in the worst case to narrow our space down to a single solution.

2. Define INDEPENDENT-SET as the problem that takes a graph \( G \) and an integer \( k \) and asks whether \( G \) contains an independent set of size \( k \). That is, \( G \) contains a set \( I \) of vertices of size \( k \) such that, for any \( v, w \in I \), there is no edge \((v, w)\) in \( G \). Show that INDEPENDENT-SET is NP-complete.

First, note that this problem is in NP, because we can guess an independent set of size \( k \) and check it in polynomial time.

To show that it is NP-hard, we will reduce from VERTEX-COVER. An instance of VERTEX-COVER is a graph \( G \) and a positive integer \( k \). We accept if there is a set of \( k \) vertices such that every edge is incident to at least one of the vertices in our set.

An independent set is a set of \( k \) vertices that have no edges between them.

Note that, if \( S \) is a vertex cover of \( G \), then \( V - S \) is an independent set, because at least one vertex of every edge is contained in \( S \). Similarly, if \( S' \) is an independent set, then \( V - S' \) is a vertex cover.

So our reduction will take \( G \) and \( k \) and produce the same graph \( G \), and the integer \( n - k \). (This obviously takes polynomial time.) There is a vertex cover of size \( k \) if and only if there
is an independent set of size \( n - k \). Therefore, INDEPENDENT-SET is NP-complete.

3. Define HYPER-COMMUNITY to be the problem that takes a collection of \( n \) web pages and an integer \( k \), and determines if there are \( k \) web pages that all contain hyperlinks to each other. Show that HYPER-COMMUNITY is NP-complete.

First, note that HYPER-COMMUNITY is in NP, since a nondeterministic machine could simply guess \( k \) web pages, and check that they are all connected to one another.

Next, to show that HYPER-COMMUNITY is NP-hard, we reduce from INDEPENDENT-SET. Suppose that we have a graph \( G \) with \( n \) vertices, and we want to find an independent set of size \( k \). We construct \( G' \) on the same \( n \) vertices, where \((v, w)\) is an edge in \( G' \) if and only if it is not an edge in \( G \). This reduction obviously takes polynomial time, since we only have to iterate over all pairs of vertices.

Now if there is a set of \( k \) mutually connected vertices in \( G' \), then they must form an independent set in \( G \). Conversely, if there is an independent set of size \( k \) in \( G \), then those \( k \) vertices must all be connected in \( G' \).

Since INDEPENDENT-SET reduces to HYPER-COMMUNITY, HYPER-COMMUNITY must be NP-complete. QED.

4. Show that the HAMILTONIAN-CYCLE problem on undirected graphs is NP-complete. The HAMILTONIAN-CYCLE problem is to determine if a given graph, \( G \), contains a cycle that visits every vertex in \( G \) exactly once.

First, note that HAMILTONIAN-CYCLE on undirected graphs is in NP, since a nondeterministic machine could guess an ordering of our vertices and then check in polynomial time that it forms a cycle.

Next, to show that the problem is NP-hard, we reduce from HAMILTONIAN-CYCLE for directed graphs. Suppose we have a directed graph \( G \) with \( n \) vertices, \( v_1, v_2, \ldots, v_n \). We construct an undirected graph \( G' \) with \( 2n \) vertices, which we label as \( x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_n, y_n, z_n \). We connect these vertices as follows.

First, note that we have created three vertices for each vertex in \( G \). We will add the edges \((x_i, y_i)\) and \((y_i, z_i)\) for all \( i \).

Next, we let \( x_i \) contain all of the outgoing edges from \( v_i \) in \( G \), and let \( z_i \) contain all of the incoming edges to \( v_i \). That is, if \((v_i, v_j)\) is an edge in \( G \), then \( x_i, z_j \) is an edge in \( G' \).

Now we show that there is a cycle in \( G' \) if and only if there was a cycle in \( G \). First, suppose that we have a cycle in \( G \), \( v_{i_1}, v_{i_2}, \ldots, v_{i_k}, v_{i_1} \). The image of these edges in \( G' \) will be the set of edges \( x_{i_1}, z_{i_k+1} \). We then have the following cycle in \( G' \):

\[
x_{i_1}, z_{i_2}, y_{i_2}, x_{i_2}, z_{i_3}, \ldots, x_{i_k}, z_{i_1}, y_{i_1}, x_{i_1}
\]

Next, suppose that we have a cycle in \( G' \). The only way that a cycle can include \( y_i \) is to use the edges \((x_i, y_i)\) and \((y_i, z_i)\). So these must be part of the cycle. Then each \( x_i \) must have
a single edge to some $z_j$. Therefore, we can find a cycle in the original graph, by using the edge from $v_i$ to $v_j$ if $x_i$ has an edge to $z_j$.

This completes all of the necessary parts of our reduction, so we know that finding a Hamiltonian cycle in an undirected graph is NP-complete. QED.

**Side note:** It is tempting to solve this problem by including only the $x$ and $z$ vertices, for outgoing and incoming edges. The first direction of the reduction will work - if $G$ has a Hamiltonian cycle, then $G'$ will as well. But this can create spurious solutions, in which $G'$ has a Hamiltonian cycle, but $G$ does not.

For example, consider the following directed graph:

This graph has no Hamiltonian cycle. But if we leave out the $y$ vertices in our construction, our new graph $G'$ will be as follows:

This graph has the cycle $x_1, z_1, x_2, z_4, x_4, z_3, x_3, z_2, x_1$. But note that we are not using the edge $(x_2, z_2)$ as we should. Thus if we include the $y$ vertices in our construction, $y_2$ cannot be included in the cycle.
5. Suppose a friend of yours is rushing for one of the university fraternities, Tau Nu Tau (TNT). His job for this week is to arrange all the bottles in the TNT beer bottle collection in a circle, subject to the constraint that each pair of consecutive bottles must be for beers that were both drunk in some TNT party. He has been given a listing of the beers in the TNT beer bottle collection, and, for each beer on the list, he is told which other beers were drunk along with this one at some TNT party. Politely show that your friend has been asked to solve an NP-complete problem.

First, note that this problem is in NP, because if there is a valid arrangement we can easily guess and check it in polynomial time.

To prove that this problem is NP-hard, we will show that it is equivalent to finding an undirected Hamiltonian cycle in a graph. For a given graph $G$, take a different beer bottle $b_i$ for each vertex $v_i$. Then if there is an edge $(v_i, v_j) \in G$, add a party with the bottles $b_i$ and $b_j$.

Now a listing of the beers for which each pair of consecutive bottles were drunk together is equivalent to a Hamiltonian path in $G$. If there is no such listing, then there is no Hamiltonian path in $G$.

Since we can reduce from Hamiltonian path to our beer bottle problem, we know that the beer bottle problem is NP-complete.