[Overall Information]
In today’s lecture, we discussed the reduction from 3SAT to INDEPENDENT SET.

[Announcements]
1. The solution to homework#5 was distributed after today’s class. This homework set is directly related to the coming quiz.
2. Quiz#5 will take place this Wednesday from 8:30am. Just like before, it will be a close-book close-note quiz which lasts around 20 minutes.
[During the Lecture]

1. At the beginning of today’s lecture, we reviewed and rewrote the INDEPENDENT SET problem. Our goal is to prove that INDEPENDENT SET is NP-complete.

There are two major steps to achieve this goal:

(1) The first step, which has already been done in a previous lecture, is to show that the INDEPENDENT SET problem belongs to NP.

(2) For the second step, in order to prove INDEPENDENT SET (as well as other problems X) NP-hard, we will use the transitivity of polynomial time reductions: Because SAT is NP-hard, each problem X reduced to SAT. And because SAT reduces to 3-SAT, and 3-SAT to INDEPENDENT SET (which we will prove), we get that for all problems X in NP, $X \leq_p SAT \leq_p 3$-SAT $\leq_p$ INDEPENDENT SET.

Theorem: IS is NP-complete

1. IS $\in$ NP, already proved
2. Every $X \in$ NP $\leq_p$ IS

Prove that 3SAT $\leq_p$ IS. We know that all X areP reduce to SAT (aka. Theorem), and SAT $\leq_p$ 3SAT by transitivity

$X \leq_p SAT \leq_p 3$-SAT $\leq_p$ IS.
Then, all problems X are known to reduce to INDEPENDENT SET, so it must be NP-hard.

2. The missing link in the above chain is that 3-SAT \( \leq_p \) INDEPENDENT SET. The reduction has to take a Boolean formula \( \Phi \) as input, and output a graph \( G \) and a number \( k \), so that:

(1) If \( \Phi \) is satisfiable, then \( G \) has an independent set of size (at least) \( k \).

(2) If \( \Phi \) is not satisfiable, then \( G \) doesn’t have an independent set of size (at least) \( k \).

3. As we can see, this time the reduction relationship is not as obvious as before (flipping the nodes or edges…). To work it out, first we had a look at a specific Boolean formula \( \Phi \) as the following picture.

A possible variable assignment which makes the formula true was given. To restate what we already know, a formula is satisfiable if there is a
variable assignment that makes at least one literal in each clause true. It is not satisfiable if for each variable assignment, at least one clause has no true literal. The question is how we can somehow “model” the decisions about which variables to set to true/false as decisions about which nodes to include in our independent set.

For the INDEPENDENT SET problem, we want to include as many nodes as possible as long as they are independent of each other, while for the 3-SAT problem with an input formula of n clauses, we would like to maximize the number of satisfiable clauses (and in particular, make the number be m, the total number of clauses).

The above two observations and analysis gave us some clues about how to design the reduction we need.
We want to design a graph so that including a node in S corresponds to making a literal true, and satisfying a clause with it. Thus, we start by having three nodes \( v(j,1), v(j,2), v(j,3) \) for each clause \( C(j) \). Since we want to count the number of satisfied clauses, and not double-count a clause just because it has multiple true literals in it, we connect the three pairs \( (v(j,1), v(j,2)), (v(j,2), v(j,3)), \) and \( (v(j,3), v(j,1)) \) by edges.

In addition, we must make sure that no two nodes can be picked together if the corresponding truth setting would be invalid. It would be invalid if it were to make both “\( x_i \)” and “\( \neg x_i \)” true. To keep the independent set from including any such pairs, we add a “conflict” edge between any two nodes \( v(j,i) \) and \( v(j',i') \) such that \( v(j,i) \) and \( v(j',i') \) are opposite literals of the same variable.

This describes the construction of the graph. The input number \( k \) (size of the independent set we are seeking) is set to \( m \), the number of clauses.

The general algorithm of this reduction was written on the whiteboard afterwards. For an input formula with \( n \) clauses, creating all the corresponding nodes takes \( O(3n) \) time. Adding conflict edges takes \( O(n^2) \) time while adding edges other than conflict edges takes \( O(n) \) time. So the total running time is polynomial \( (O(n^2)) \).
Next, we had to show that the reduction is correct. The proof very much followed the intuition that we used in coming up with the reduction in the first place.

Part I:

Suppose an input formula $\Phi$ is satisfiable, and we know one variable assignment which makes $\Phi$ true. Then, for each clause of $\Phi$, at least one literal must be true under the assignment. In the graph $G$ produced by the above reduction, we are able to pick one node corresponding to a true literal for each clause (if there are multiple choices, pick an arbitrary one). Let $S$ be the selected set.

Because only one node corresponding to each clause is picked, none of the “triangle edges” for clauses are inside the selected set $S$. Also, no conflict edge can ever be inside $S$, because that would require us to pick two conflicting nodes (for example, the nodes corresponding to “$x_1$” and “not $x_1$”); but the corresponding two literals will never be true
simultaneously. So far, we have proved that if $\Phi$ is satisfiable, then $G$ has an independent set of size $k=m$.

6. Part II:

In this part, we will prove that if $G$ has an independent set of size at least $k = m$, then $\Phi$ is satisfiable.

Given an independent set $S$ of size $m$ in the graph $G$, we first observe that for each clause $j$, $S$ contains exactly one node from among the nodes corresponding to literals of the clause. The reason is that the triangle edges prohibit $S$ from containing two nodes, and if any triangle had no node at all in $S$, then the total could not add up to $m$. We set the literals corresponding to the node $v(i,j)$ in $S$ to true. After that, all the literals not assigned yet can have arbitrary true/false values.
(1) The existence of conflict edges guarantees that each variable is either set to true or false this way, but not both. As a result, the assignment is valid (or, as it is called correctly, “well-defined”).

(2) Since for every group of nodes corresponding to a clause of \( \Phi \), at least one node is in \( S \) and consequently, the corresponding literal in the Boolean formula is set to true, this assignment indeed makes the whole formula true.

So we have shown that if \( G \) has an independent set of size \( k \), then \( \Phi \) is satisfiable. This is equivalent to saying that “if \( \Phi \) is not satisfiable, then \( G \) doesn’t have an independent set of size \( k \)”. This completes the
In reflecting on the proof, notice what we did. We found a way to “encode” the decision of whether to make a variable true or false in the decision whether to include a node in an independent set or not. We used the edges in the graph as constraints to make sure that the node sets picked somehow mimicked the constraints that the formula imposed on the variables. The intuition that we used in designing the reduction then guided the correctness proof.