1 Summary of Material

1.1 P vs. NP Definitions

There are 2 classes of problems that we will examine in this course: **P** and **NP**. In today's section, we will review the difference between the two, as well as get some more practice classifying something as **P**, **NP**, and **NP**-Complete.

- **P** - the set of all yes/no problems that can be deterministically solved in polynomial time, that is runs in $O(n^k)$ steps for some positive integer k.
- **NP** - the set of all yes/no problems where one could convince you that the answer is "yes" by giving a polynomial length certificate which can be verified in polynomial time as well.
  - **Examples from Lecture:** Compositeness (Certificate = the factors, Checking = multiplication), 3-SAT (Certificate = the satisfying assignment, Checking = evaluate it)

Note that P is a subclass of NP because any we can verify that a P algorithm is correct by simply running it, which takes polynomial time.

1.2 Reductions

A reduction is a procedure $R(x)$ which transforms the input for problem A into an input for problem B. This transformation must maintain the integrity of the answer, that is the answer to A is yes from an input x if and only if the answer to B from $R(x)$ is yes. If we can find such a procedure $R(x)$, then we can conclude that A is at least as easy as B. This is true because we know that **if we can solve B, we can solve A** by making this reduction.

To Show that A is Easy: Reduce A to something easy.

To Show that A is Hard: Reduce something else that is hard to it.

1.3 NP-Complete

These are the hardest problems in NP, which have the property that all other problems in NP reduce to them. Our time-line of proving various problems to be NP-Complete is summarized below:

1. **Circuit-SAT**: Cook's Theorem (informal explanation offered in lecture)
2. **3-SAT**: Circuit-SAT reduced to 3-SAT by introducing various combinations of 3 clauses to represent x being an AND, OR, or NOT gate of y and z.
3. **Integer Linear Programming**: 3-SAT reduced to this by replacing each literal with x or $1 - x$ and then setting the sum of each clauses to be greater than 1.
4. **Independent Set**: 3-SAT reduced to this by constructing a strange graph where nodes represented assignments of the literals, and edges connected assignments that conflicted with each other.

5. **Vertex Cover:** : Independent Set and Vertex Cover reduce to each other by observing that C is an independent set if and only if $V - C$ is a vertex cover.

2 Practice Problems

Exercise. *Consider the Set Cover problem: Given a set U of elements and a collection $S_1, ..., S_m$ of subsets of U, is there a collection of at most k of these sets whose union equals U? You may recall that there is a greedy algorithm which is off by a factor of $O(\log n)$. Show that Set Cover is actually NP-Complete by reducing one of the 5 NP-Complete algorithms above to it.*

Solution.
We will show that Vertex Cover reduces to Set Cover by (1) Explaining how to convert an instance of VC into SC in polynomial time, and (2) showing that the two problems become equivalent (i.e. "yes" in VC if and only if "yes" in SC).

Given an instance of vertex cover with (V, E, k), we construct our universe U to be the set of all edges in E and our sets S_u to be the edges adjacent to the vertex $u \in V$.

Proof. If there exists a valid vertex cover, then the corresponding set cover by choosing the sets constructed from the edges of the corresponding vertices in the vertex cover must form a set cover. This is true by construction because if all the edges have been covered in E, then all the items have been covered in U.

On the other hand, if there exists a valid set cover, then we can choose the vertices that are represented by each set. Since each edge corresponded to at least 1 set, each edge in the graph will have at least one neighbor that was chosen.

Exercise. *The Subset Sum problem is defined as follows: Given a sequence of integers $a_1 \dots a_n$ and a parameter k, decide whether there is a subset of the integers whose sum is exactly k. Show Subset Sum is NP-complete by reduction from Vertex Cover*

Solution.
We prove that Subset Sum is NP-complete by reduction from Vertex Cover. We have to proceed as follows:

1. Start from a graph G and a parameter k.
2. Create a sequence of integers and a parameter k'.
3. Prove that the graph has vertex cover with k vertices if and only if there is a subset of the integers that sum to k'

Consider the following table which has $n + m$ rows and $m + 1$ columns. We fill it in as follows:

- For the 0th column, place a 1 in the first n rows, and a 0 everywhere else.
- For column $i > 0$, take the ith edge $e_i = (v_a, v_b)$ and mark the a, bth entires from the first section (indexed 1 to n) and also mark the ith entry from the second section (indexed 1 to m).

	0	1	2	...	$m-1$	m	Number
1	1						?
2	1						?
$\vdots$	1						?
n	1						?
1	0	1					4^{m-1}
2	0		1				4^{m-2}
$\vdots$	0						
$m-1$	0				1		4
m	0					1	1

After filling in the table above, we define each number to be base 10 value of each row when treated as a number in base 4. The digits will be binary, but we will still convert them to base 4. This will become clear later on.

Now, we take

$$k' = k \cdot 4^m + \sum_{i=0}^{m-1} 2 \cdot 4^i$$

and the claim is that having a vertex cover of size k is equivalent to having a subset sum of k'.

Proof. Suppose we have a vertex cover. Then, we first choose the k rows from the first n which correspond to the vertex cover. By the definition of vertex cover, each column 1 throug m must have at least one 1 contributed by those k rows. Each column can contain one or two 1's from these k rows. For the columns that have only one 1, we also choose the corresponding column from the last m rows. This guarantees that each column 1 through m has exactly two 1's in it, so when we compute the sum, we get the desired k'.

Suppose that there is a subset of the integers such that the sum is exactly k'. Then, we know that there must be exactly k integers chosen from the first n because $4^m \cdot k$ cannot be otherwise. By the way we constructed everything, no "carrying" can ever occur because each row has only three 1's. Therefore, the only way for k 4^m's to occur is to choose k integers from the first n. Now, there may also be integers chosen from the last m rows. Collectively, the k rows from the first n and some subset of the last m rows must contribute at least two 1's to each column. Since out of the last m rows, there is only one 1 per column, the only way to get two one's is to have a contribution in each column from one of the first k rows. Therefore, by construction, each edge is covered by one of the k vertices we chose from the first n integers.