• **DISCLAIMER:** These notes are not necessarily an accurate representation of what I said during the class. They are mostly what I intend to say, and have not been carefully edited.

• Main topics:
  - Return test 4.
  - NP-hardness

• Test 4:
  - Question 3c: :-(.
  - Linear programs: standard form was not covered in class, will rerun at a later date.

• Grade distribution so far:
  - $20 \times 3 + 3 \times 4 = 72$ points out so far, 3 + 25 to go.
  - Can compute grade by adding together points on top 3 tests, then divide homework total by 5.
  - Ballpark cutoffs: $A \approx 60$, $B \approx 45$, $C \approx 35$.

In the last part of this course, we will discuss problems that are difficult in the theoretical sense. So far one of the bars that we’ve been using to describe an algorithm as efficient is polynomial time: $O(n^c)$ for some constant $c$.

To formally define classes of problems, it’s simpler to consider decision problems: instead of asking of the length of the shortest path, we can instead ask the question of whether there exists a path of length at most $x$. Formally, we can consider the input as provided by a string of length $n$, and use $n$ to denote the input size. A decision problem $\Pi$ can be viewed as a problem that assigns every input, $x$, a value that’s true and false. In this view, a polynomial time algorithm can be viewed as:

**Definition 0.1.** A decision problem $\Pi$ is in P if there is a polynomial time verification algorithm $V_\Pi$ such that $\Pi(x) = true$ if and only if $V_\Pi(x) = true$. 
Almost every problem that we’ve discussed so far fall into this category. On the other hand, for some of the problems, it’s easier to verify a solution than computing an answer. A good example of this is longest path: given a path, we can check whether it repeats vertices, and whether its length is greater than some threshold $x$. This leads the notion of non-deterministic polynomial time, or NP, where a solution can be verified in polynomial time.

**Definition 0.2.** A decision problem $\Pi$ is in NP if there exists a polynomial time verification algorithm $V_\Pi$ and a constant $c > 0$ such that $\Pi(x) = true$ if and only if there exists a certification $y$ such that:

- $|y| \leq |x|^c$, and
- $V_\Pi(x, y) = true$.

A good example of a problem in NP is minimum vertex here. Here the decision version is whether there exists a vertex cover of size at most $k$.

To show that this problem is in $NP$, we can have let the certificate be the vertices that are in the cover. This has length $|V|$, which is at most the input size. Given such a string, we can then check in linear time whether it is a vertex cover, and whether its size is at most $k$.

It can also be checked that any problem in P is also in NP: the verifier can simply take the empty string. On the other hand, anything in NP can be solved in exponential time by trying all $2^n^c$ certificates $y$ and check if the verifier returns true.

NP contains a much larger list of problems, including, maximum independent set, longest path, integer programming, and satisfiability.