Precept Outline

- Review of Lectures 23 and 24:
  - Intractability
  - Algorithm Design and Final Review

A. Review: Intractability

Your preceptor will briefly review key points of this week's lectures.

B. Problem solving: Poly-time Reductions

In this problem, we will prove a well-known intractability result: that 3-SATISFIABILITY (3-SAT hereafter) is NP-complete. Before we do so, let's see some important definitions:

SAT: In the SATISFIABILITY (SAT hereafter) problem we are given a system of m boolean equations in n variables as input. The goal is to determine whether there is an assignment to the variables that satisfies all equations. Here is an example of a SAT instance with m = 3 and $n = 4$:

$$\begin{aligned} x_1 \text{ or } x_2 \text{ or } x_3 &= \text{ true} \\ \neg x_1 \text{ or } \neg x_3 &= \text{ true} \\ x_4 \text{ or } \neg x_2 \text{ or } x_3 \text{ or } x_1 &= \text{ true} \end{aligned}$$

If we set $x_1 =$ false, $x_2 =$ true, $x_3 =$ false, and $x_4 =$ true, then all of the equations are satisfied, so this is a satisfiable instance of SAT.

3-SAT: In the 3-SAT problem we are given a system of m boolean equations each containing exactly 3 literals (a literal is either a variable or its negation) in n variables as input. The goal is to determine whether is there an assignment to the variables that satisfies all equations. Note that the example above is **not** a valid instance of 3-SAT since the second equation only has 2 literals and the last one has 4.

To show that 3-SAT is NP-complete we need to prove two separate things:

- 3-SAT is in NP (i.e., there is a polynomial-time algorithm for verifying a candidate solution);
- All problems in NP poly-time reduce to 3-SAT.

Start by showing that 3-SAT is in NP by describing a polynomial-time algorithm for verifying a candidate solution.

To show that all problems in NP poly-time reduce to 3-SAT, first recall that in lecture we saw that SAT is NP-complete, so if we find a poly-time reduction from SAT to 3-SAT then we accomplish this.

In order to do so, we can try to construct (in polynomial time) an instance $\mathcal{I}_3$ of 3-SAT from an instance $\mathcal{I}_S$ of SAT such that $\mathcal{I}_3$ is satisfiable if and only if $\mathcal{I}_S$ is satisfiable. Note that $\mathcal{I}_3$ and $\mathcal{I}_S$ can have different sizes (so different number of variables and/or equations), as long as one can construct $\mathcal{I}_3$ from $\mathcal{I}_S$ in polynomial time.

Given an instance $\mathcal{I}_S$ of SAT, describe a polynomial time algorithm to find some instance $\mathcal{I}_3$ of 3-SAT with the above properties.

Apply the above transformation to the example in the beginning of this section.

C. Problem solving: (Final Review): Detecting Biased Dice

Suppose you have k 6-sided uneven dice. Because they are uneven, each outcome has a different probability p_i, which is given to you. These are always valid probabilities, meaning $p_1 + p_2 + p_3 + p_4 + p_5 + p_6 = 1$ and they are all non-negative numbers (but some could be 0!).

Describe an algorithm that finds the maximum possible sum of the outcomes obtainable by rolling all k dice, as well as the probability of obtaining that sum. The running time of your algorithm should be $\Theta(k)$.

In the space provided, give a concise English description of your algorithm for solving the problem. You may use any of the algorithms that we have considered in this course (e.g., lectures, precepts, textbook, assignments) as subroutines. If you modify such an algorithm, be sure to describe the modification. Feel free to use code or pseudocode to improve clarity.

Given an integer X, your task is to determine what is the probability that the sum of the outcomes of rolling the k dice is exactly X. Describe an algorithm to do so with running time of $\Theta(X \cdot k)$.

In the space provided, give a concise English description of your algorithm for solving the problem. You may use any of the algorithms that we have considered in this course (e.g., lectures, precepts, textbook, assignments) as subroutines. If you modify such an algorithm, be sure to describe the modification. Feel free to use code or pseudocode to improve clarity.