CS41 Lab 8: Polynomial-Time Verifiers

This week, we’ve started to understand what makes some problems seemingly hard to compute. In this lab, we’ll consider an easier problem of verifying that an algorithm’s answer is correct.

1. Reducing Vertex-Cover to Independent-Set. Recall from class the following problems
   - **Independent-Set** takes an undirected graph \( G = (V, E) \) and integer \( k \) and returns yes iff \( G \) contains an independent set of size at least \( k \).
   - **Vertex-Cover** takes an undirected graph \( G = (V, E) \) and integer \( k \) and returns yes iff \( G \) contains a vertex cover of size at most \( k \).

   (a) Show that **Independent-Set** \( \leq_P \) **Vertex-Cover**.
   (b) Show that **Vertex-Cover** \( \leq_P \) **Independent-Set**.

2. Polynomial-time Verifiers. Call \( V \) an efficient **verifier** for a decision problem \( L \) if
   (a) \( V \) is a polynomial-time algorithm that takes two inputs \( x \) and \( w \).
   (b) There is a polynomial function \( p \) such that for all strings \( x \), \( x \in L \) if and only if there exists \( w \) such that \( |w| \leq p(|x|) \) and \( V(x, w) = yes \).

   \( w \) is usually called the **witness** or **certificate**. Think of \( w \) as some proof that \( x \in L \). For \( V \) to be a polynomial-time verifier, \( w \) must have size some polynomial of the input \( x \). For example, if \( x \) represents a graph with \( n \) vertices and \( m \) edges, the length of \( w \) could be \( n^2 \) or \( m^3 \) or \( (n + m)^{100} \) but not \( 2^n \).

   Give polynomial-time verifiers for the following problems which are not known to have efficient
   (a) **Independent-Set**.
   (b) **Vertex-Cover**.
   (c) **Three-Coloring**. Given \( G = (V, E) \) return yes iff the vertices in \( G \) can be colored using at most three colors such that each edge \((u, v) \in E\) is bichromatic.
   (d) **Wedding-Planner**. Recall in the Wedding planner problem, the input consists of a list of \( n \) people to possibly invite to a wedding, along with \( m \) **clauses**, where each clause specifies some criteria for whom to invite or not invite. Output yes iff there exists an invitation list that satisfies all clauses.
   **Note:** Assume that each clause is of the form e.g. \( x_1 \lor \bar{x}_2 \lor \cdots \lor \bar{x}_k \), where \( x_i \) means to invite person \( i \), and \( \bar{x}_j \) means to not invite person \( x_j \).
   (e) **Factoring**. Given numbers \( n, k \) written in binary, output yes iff \( n \) is divisible by \( d \) for some \( 1 < d \leq k \).
   (f) **Not-Factoring**. Given numbers \( n, k \) written in binary, output yes iff \( n \) is not divisible by \( d \) for any \( 1 < d \leq k \).

   **Hint:** The following problem is solvable in polynomial time.

   **Primes:** Given a number \( n \) written in binary, output yes iff \( n \) is a prime number.

---

1This actually wasn’t known until 2002, when Agrawal, Kayal, and Saxena created the AKS primality test. Kayal and Saxena were undergraduates at IIT Kanpur at the time; Agrawal was their advisor.