Topic: Reductions (using reachability). How can we use reachability algorithms to solve new problems?

Helpful Resources: Reductions: Using reductions between problems to design algorithms and Graph Search Algorithms: Design and Correctness

1. The Graph Reachability Problem:
   (a) Setup: Let \( G = (V,E) \) be a graph, and let \( s \) and \( t \) be two distinct vertices in \( V \).
   (b) Output: Is there a path from \( s \) to \( t \) using edges in \( E \)?
   (c) Some Algorithms for Reachability:
      i. Breadth-first search (BFS), Depth-first search (DFS), Dijkstra’s algorithm
      ii. BFS and DFS solve reachability in time \( O(|V| + |E|) \).
      iii. Dijkstra’s algorithm finds shortest path between two nodes.
      iv. Dijkstra’s algorithm can solve reachability in time \( O(|V|^2) \)
         (potentially faster using specialized data structures and for special cases).

2. Two main approaches to use known algorithms to create new algorithms.
   • Black-box: Use known algorithms as subroutines (“reductions”)
   • Modification: Tweak an existing algorithm (“white-box”)

3. Decision Problems
   (a) A decision problem has “Yes” /“No” answers.
   (b) Let \( \Pi \) be a decision problem. Let \( x \) be an instance. We say \( x \in \Pi \) if \( x \) is a “Yes” instance of \( \Pi \).
   (c) Note: We generally define decision problems by their “Yes” instances.
   (d) Example: \( \Pi = \text{Graph Reachability} \). \( x = (G,s,t) \). We say \((G,s,t) \in \text{Graph Reachability}\) if there is a path from \( s \) to \( t \) in \( G \).

4. Reduction outline:
   (a) Given: An instance \( x_1 \) and a decision problem \( \Pi_1 \).
   (b) Task: Is \( x_1 \in \Pi_1 \)?
   (c) Know: An efficient algorithm \( Alg_2 \) for \( \Pi_2 \).
      (For today, imagine \( \Pi_2 \) is Graph Reachability and \( Alg_2 \) is one of the above reachability algorithms.)
   (d) Approach: Construct a reduction \( F \). Let \( x_2 := F(x_1) \). Return \( Alg_2(x_2) \).
   (e) Correctness analysis: Need \( F \) to map “Yes” instances to “Yes” instances and “No” instances to “No instances”.
      • If \( x_1 \in \Pi_1 \), then \( x_2 \in \Pi_2 \), and
      • If \( x_2 \in \Pi_2 \), then \( x_1 \in \Pi_1 \) (or equivalently, if \( x_1 \notin \Pi_1 \), then \( x_2 \notin \Pi_2 \)).

5. In the lectures, we learned about how graph reachability could be used to solve the max bandwidth problem. How could we use reachability to solve the following problems?
   (a) Given a graph \( G = (V,E) \) and two vertices \( s \) and \( t \), where each edge is labeled either 0 or 1, is there a path from \( s \) to \( t \) with an even number of 1’s?
   (b) Given a graph \( G = (V,E) \) and two vertices \( s \) and \( t \), where each edge is labeled either 0 or 1, is there a path from \( s \) to \( t \) where all 0s come before all 1s?
   (c) Given a graph \( G = (V,E) \) and two vertices \( s \) and \( t \), where each edge is given a weight \( w(e) \), is there a path from \( s \) to \( t \) where the weights are strictly decreasing?
(d) Given a DAG $G = (V, E)$ and two vertices $s$ and $t$, where each edge is given a label $\ell(e)$, is there a path (not necessarily simple) from $s$ to $t$ where the concatenation of the labels is a palindrome? (A palindrome is a string which is the same forwards or backwards.)

6. A $k$-CNF formula is a boolean function written as a conjunction of disjunctions, called clauses, each containing at most $k$ literals (a variable or its negation). For example, $(x \lor y) \land (\neg x \lor \neg y)$ is a 2-CNF which computes whether exactly one of $x$ or $y$ is true. The $k$-SAT problem is the problem of checking whether a given $k$-CNF formula is satisfiable; that is, whether there is some way of assigning true or false to each variable, such that the entire formula evaluates to true.

It is well known that the 3-SAT problem is $\text{NP}$-complete, and has no known efficient algorithms (in fact, it’s suspected that it requires exponential time in the worst case). However, 2-SAT has linear time algorithms, that depend on computing the strongly connected components of a particular graph.

We can think of a 2-SAT formula $\phi$ as a collection of implications; each clause $(x \lor y)$ can be seen as $\neg x \Rightarrow y$ or $\neg y \Rightarrow x$. Consider the directed graph $G_{\phi}$ where, for each variable $x$ in $\phi$, we have a vertex labeled $x$ and another labeled $\neg x$, and for each clause $(l_1 \lor l_2)$ we have an edge from $\neg l_1$ to $l_2$ and another from $\neg l_2$ to $l_1$.

(a) Prove that for any truth assignment which satisfies $\phi$, all literals in the same strongly connected component are assigned the same truth value.

(b) Prove that if for some variable $x$, $x$ and $\neg x$ are in the same strongly connected component, then $\phi$ is unsatisfiable.

(c) Prove that if for all variables $x$, $x$ and $\neg x$ are in different strongly connected components, then $\phi$ is satisfiable. (Hint: think about what an edge means in the SCC DAG, and which variables you would assign values to first.)

7. Remaining Time: Discuss Homework-01 and Quiz 1