CS 170 Homework 10

Due **4/11/2022, at 10:00 pm (grace period until 11:59pm)**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write “none”.

2 A Reduction Warm-up

In the Undirected Rudrata path problem (aka the Hamiltonian Path Problem), we are given a graph G with undirected edges as input and want to determine if there exists a path in G that uses every vertex exactly once.

In the Longest Path in a DAG, we are given a DAG, and a variable k as input and want to determine if there exists a path in the DAG that is of length k or more.

Is the following reduction correct? Please justify your answer.

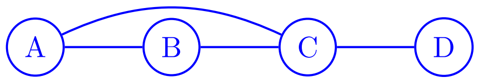
Undirected Rudrata Path can be reduced to Longest Path in a DAG. Given the undirected graph G, we will use DFS to find a traversal of G and assign directions to all the edges in G based on this traversal. In other words, the edges will point in the same direction they were traversed and back edges will be omitted, giving us a DAG. If the longest path in this DAG has $|V| - 1$ edges then there must be a Rudrata path in G since any simple path with $|V| - 1$ edges must visit every vertex, so if this is true, we can say there exists a Rudrata path in the original graph. Since running DFS takes polynomial time ($O(|V| + |E|)$), this reduction is valid.

Solution:

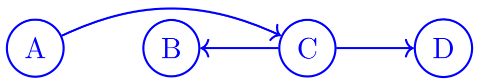
It is incorrect.

It is true that if the longest path in the DAG has length $|V| - 1$ then there is a Rudrata path in G. However, to prove a reduction correct, **you have to prove both directions**. That is, if you have reduced problem A to problem B by transforming instance I to instance I' then you should prove that I has a solution **if and only if** I' has a solution. In the above ”reduction,” one direction does not hold. Specifically, if G has a Rudrata path then the DAG that we produce does not necessarily have a path of length $|V| - 1$–it depends on how we choose directions for the edges.

For a concrete counterexample, consider the following graph:



It is possible that when traversing this graph by DFS, node C will be encountered before node B and thus the DAG produced will be



which does not have a path of length 3 even though the original graph did have a Rudrata path.

3 Decision vs. Search vs. Optimization

Recall that a vertex cover is a set of vertices in a graph such that every edge is adjacent to at least one vertex in this set.

The following are three formulations of the VERTEX COVER problem:

- As a *decision problem*: Given a graph G, return TRUE if it has a vertex cover of size at most b, and FALSE otherwise.
- As a *search problem*: Given a graph G, find a vertex cover of size at most b (that is, return the actual vertices), or report that none exists.
- As an *optimization problem*: Given a graph G, find a minimum vertex cover.

At first glance, it may seem that search should be harder than decision, and that optimization should be even harder. We will show that if any one can be solved in polynomial time, so can the others.

(a) Suppose you are handed a black box that solves VERTEX COVER (DECISION) in polynomial time. Give an algorithm that solves VERTEX COVER (SEARCH) in polynomial time.

(b) Similarly, suppose we know how to solve VERTEX COVER (SEARCH) in polynomial time. Give an algorithm that solves VERTEX COVER (OPTIMIZATION) in polynomial time.

Solution:

(a) If given a graph G and budget b, we first run the DECISION algorithm on instance (G, b). If it returns "FALSE", then report "no solution".

If it comes up "TRUE", then there is a solution and we find it as follows:

- Pick any node $v \in G$ and remove it, along with any incident edges.
- Run DECISION on the instance $(G \setminus \{v\}, b - 1)$; if it says "TRUE", add v to the vertex cover. Otherwise, put v and its edges back into G.
- Repeat until G is empty.

Correctness: If there is no solution, obviously we report as such. If there is, then our algorithm tests individual nodes to see if they are in any vertex cover of size b (there may be multiple). If and only if it is, the subgraph $G \setminus \{v\}$ must have a vertex cover no larger than $b - 1$. Apply this argument inductively.

Running time: We may test each vertex once before finding a v that is part of the b-vertex cover and recursing. Thus we call the DECISION procedure $O(n^2)$ times. This can be tightened to $O(n)$ by not considering any vertex twice. Since a call to DECISION costs polynomial time, we have polynomial complexity overall.

Note: this reduction can be thought of as a greedy algorithm, in which we discover (or eliminate) one vertex at a time.

(b) Binary search on the size, b, of the vertex cover.

Correctness: This algorithm is correct for the same reason as binary search.

Running time: The minimum vertex cover is certainly of size at least 1 (for a nonempty graph) and at most $|V|$, so the SEARCH black box will be called $O(\log |V|)$ times, giving polynomial complexity overall.

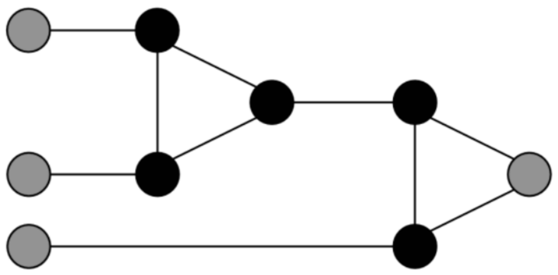
Finally, since solving the optimization problem allows us to answer the decision problem (think about why), we see that all three reduce to one another!

4 Reduction to 3-Coloring

Given a graph $G = (V, E)$, a valid 3-coloring assigns each vertex in the graph a color from $\{0, 1, 2\}$ such that for any edge (u, v), u and v have different colors. In the 3-coloring problem, our goal is to find a valid 3-coloring if one exists. In this problem, we will give a reduction from 3-SAT to the 3-coloring problem. Since we know that 3-SAT is NP-Hard (there is a reduction to 3-SAT from every NP problem), this will show that 3-coloring is NP-Hard (there is a reduction to 3-coloring from every NP problem).

In our reduction, the graph will start with three special vertices, labelled "True", "False", and "Base", and the edges (True, False), (True, Base), and (False, Base).

(a) Consider the following graph, which we will call a "gadget":



Show that in any valid 3-coloring of this graph which does not assign the color 2 to any of the gray vertices, the gray vertex on the right is assigned the color 1 only if one of the gray vertices on the left is assigned the color 1.

(b) For each variable x_i in a 3-SAT formula, we will create a pair of vertices labeled x_i and $\neg x_i$. How should we add edges to the graph such that in any valid 3-coloring, one of

$x_i, \neg x_i$ is assigned the same color as True and the other is assigned the same color as False?

(c) We observe the following about the graph we are creating in the reduction:

  (i) For any vertex, if we have the edges (v, False) and (v, Base) in the graph, then in any valid 3-coloring v will be assigned the same color as True.

  (ii) Through brute force one can also show that in the gadget, for any assignment of colors to gray vertices such that:

    (1) All gray vertices are assigned the color 0 or 1

    (2) The gray vertex on the right is assigned the color 1

    (3) At least one gray vertex on the left is assigned the color 1

  Then there is a valid coloring for the black vertices in the gadget.

Using these observations and your answers to the previous parts, give a reduction from 3-SAT to 3-coloring. Prove that your reduction is correct.

Solution:

(a) It is easier to show the equivalent statement that if all the gray vertices on the left are assigned the color 0, then the gray vertex on the right must be assigned the color 0 as well. Consider the triangle on the left. Since all the gray vertices are assigned 0, the two left points must be assigned the colors 1 and 2, and so the right point in this triangle must be assigned 0 in any valid coloring. We can repeat this logic with the triangle on the right, to conclude that the gray vertex on the right must be assigned 0 in any valid coloring.

(b) We add the edges (x_i, $\neg x_i$), (x_i, Base) and ($\neg x_i$, Base). Since $x_i, \neg x_i$ are both adjacent to Base they must be assigned a different color than Base, i.e. they both are assigned either the color of True or the color of False. Since we added an edge between x_i and $\neg x_i$, they can't be assigned the same color, i.e. one is assigned the same color as True and one the same color as False.

(c) Given a 3-SAT instance, we create the three special vertices and edges described in the problem statement. As in part a, we create vertices x_i and $\neg x_i$ for each variable x_i, and add the edges we gave in the answer to part a. For clause j, we add a vertex C_j and edges (C_j, False), (C_j, Base). Lastly, for clause j we add vertices and edges to create a gadget where the three gray vertices on the left of the gadget are the vertices of three literals in the clause, and the gray vertex on the right is the vertex C_j (All black vertices in the gadget are only used in this clause's gadget).

If there is a satisfying 3-SAT assignment, then there is a valid 3-coloring in this graph as follows. Assign False the color 0, True the color 1, and Base the color 2; assign x_i the color 1 if x_i is True and 0 if x_i is False (vice-versa for $\neg x_i$). Assign each C_j the color 1. Lastly, fix any gadget. Since the 3-SAT assignment is satisfying, in each gadget at least one of the gray vertices on the left is assigned 1, so by the observation (ii) in the problem statement the gadget can be colored.

If there is a valid 3-coloring, then there is a satisfying 3-SAT assignment. By symmetry, we can assume False is colored 0, True is colored 1, and Base is colored 2. Then for each literal where x_i is color 1, that literal is true in the satisfying assignment. By part a, we know that exactly one of $x_i, \neg x_i$ is colored 1, so this produces a valid assignment. By observation (i), we also know every node C_j must be colored 1. All literal nodes are colored 0 or 1, so by part b, this implies that for every clause, one of the gray literal nodes in the clause gadget is colored 1, i.e. the clause will be satisfied in the 3-SAT assignment.