CS 170 HW 12

Due **2020-4-20, at 10:00 pm**

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 Opting for releasing your solutions

We are considering releasing a subset of homework submissions written by students for students to see what a full score submission looks like. If your homework solutions are well written, we may consider releasing your solution. If you wish that your solutions not be released, please respond to this question with a "No, do not release any submission to any problems". Otherwise, say "Yes, you may release any of my submissions to any problems".

3 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.

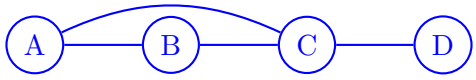
Undirected Rudrata Path can be reduced to Longest Path in a DAG. Given the undirected graph G, we will use DFS to find a traversal of G and assign directions to all the edges in G based on this traversal. In other words, the edges will point in the same direction they were traversed and back edges will 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:

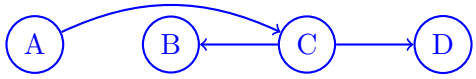
It is incorrect.

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.

4 (3,3)-SAT

Consider the (3,3)-SAT problem, which is the same as 3-SAT except each literal or its negation appears *at most* 3 times across the entire formula. Notice that (3,3)-SAT is reducible to 3-SAT because every formula that satisfies the (3,3)-SAT constraints satisfies those for 3-SAT. We are interested in the other direction.

Show that (3,3)-SAT is NP-Hard via reduction from 3-SAT. By doing so, we will have eliminated the notion that the "hardness" of 3-SAT was in the repetition of variables across the formula.

Give a precise description of the reduction and prove its correctness.

Solution:

Assume that the variable x_i or $\neg x_i$ appears $k > 3$ times. Let us replace the k occurrences of x_i with $x_i^{(j)}$ for $j = 1, \ldots, k$. We now have used each literal $x_i^{(j)}$ exactly once. We additionally add the clauses $\neg x_i^{(j)} \vee x_i^{(j+1)}$ and $\neg x_i^{(k)} \vee x_i^{(1)}$ for each $j = 1, \ldots, k-1$. Recall that $\neg x_i^{(j)} \vee x_i^{(j+1)}$ is equivalent to $x_i^{(j)} \Rightarrow x_i^{(j+1)}$ and $\neg x_i^{(j+1)} \Rightarrow \neg x_i^{(j)}$, so the collective chain enforces that $x_i^{(1)} = x_i^{(2)} = \ldots = x_i^{(k)}$.

Make the replacement for each literal that occurs more than 3 times ensures that all literals in the new formula occur at most 3 times. This is only a polynomial increase in the size of the problem description.

A satisfying assignment $\{x_i = \alpha_i\}$ for the original formula can be translated to an assignment for the new formula by setting $x_i^{(j)} = \alpha_i$ by the previous argument.

For the other direction, a satisfying assignment for the new formula must have $x_i^{(1)} = x_i^{(2)} = \ldots = x_i^{(k)} = \alpha_i$, so we can set $x_i = \alpha_i$ in the original formula.

5 Public Funds

You are looking to build a new fence for your mansion, to keep out pesky people protesting profligate purchases. You have m bank accounts at your disposal to use to pay for your fence; each account i has a balance of b_i. You must choose one of n options for your fence; each

fence j costs c_j dollars. You would like to withdraw from at most k of the bank accounts to build the fence, and due to peculiar UC accounting rules, if you use a particular bank account, you must use the whole balance (all b_m dollars.)

Determine whether it is possible to exactly pay for some fence j; that is, whether there is a j between 1 and n such that you can withdraw exactly c_j dollars given the bank account balances $b_1, \ldots, b_m$, the fence costs $c_1, \ldots, c_n$, and k, and if so return the corresponding choice of fence and set of bank accounts that you withdraw from.

Prove that Public Funds is NP-Complete. Look at problems from the textbook for possible NP-Complete problems!

Solution:

Proof that Public Funds is in NP.
We can verify a solution for Public Funds in polynomial time by verifying that the sum of the balances of the chosen bank accounts adds up exactly to the price of the fence.

Proof that Public Funds is in NP-Hard.
To show that it is in NP-Hard, we reduce from SUBSET SUM.

Let each of the m elements in the set of numbers correspond to a bank account's balance b_m. Let the desired sum s correspond to the price of the only fence c_1. Also, set the maximum number of bank accounts used, k, to the number of elements in the set of numbers m.
1. **If an instance of subset sum has a solution, then the transformed instance of Public Funds has a solution.**
If subset sum has a solution, then we can take the banks that correspond to the elements of the solution, and those banks should sum up to the only fence, thus Public Funds should have a solution.
2. **If an instance of Public Funds in the format of the transformation has a solution, then the corresponding instance of subset sum has a solution.**
If an algorithm for Public Funds finds a solution, we can interpret the bank accounts chosen as a solution for SUBSET SUM.

6 Dominating Set

A dominating set of a graph $G = (V, E)$ is a subset D of V, such that every vertex not in D is a neighbor of at least one vertex in D. Let the Minimum Dominating Set problem be the task of determining whether there is a dominating set of size $\leq k$. Show that the Minimum Dominating Set problem is NP-Complete. You may assume that G is connected.
Solution:

Proof that Minimum Dominating Set is in NP.
Given a possible solution, we can check that it is a solution by iterating through the vertices not in the solution subset D and checking if it has a neighbor in D by iterating through the adjacent edges. This would take at most E time because you would at most check every edge twice. We should also check that the number of vertices in D is less than k, which would

take at most V as k should be less than E. So, we can verify in $O(E)$ time which is polynomial.

Proof that Minimum Dominating Set is NP-Hard.
Reduction from Vertex Cover to Dominating Set
Minimum Vertex Cover is to find a vertex cover (a subset of the vertices) which is of size $\leq k$ where all edges have an endpoint in the vertex cover.

Suppose $G(V, E)$ is an instance of vertex cover and we are trying to find a vertex cover of size $\leq k$. For every edge (u, v), we will add a new vertex c and two edges (c, u), and (c, v). We will pass in the same k to the dominating set. This is a polynomial reduction because we are adding $|E|$ vertices and $2|E|$ edges.

Proof of Correctness of Reduction
1. **If minimum vertex cover has a solution, then minimum dominating set that corresponds to it has a solution.**
Suppose we have some minimum vertex cover of size k. By definition of vertex cover, every edge has either one or both endpoints in the vertex cover. Thus if we say that the vertex cover is a dominating set, every original vertex must either be in the dominating set or adjacent to it. Now we have to figure out whether the vertices we added for every edge are covered. Since every edge has to have one or both endpoint in the vertex cover, the added vertices must be adjacent to at least one vertex in the vertex cover. Thus the vertex cover maps directly to a dominating set in the transformed problem.

2. **If minimum dominating set in the transformed format has a solution, then the corresponding minimum vertex cover has a solution.**
Suppose we have some minimum dominating set of size k of the transformed format. Vertices in the dominating set either must come from the original vertices or the vertices we added. If they don't come from the vertices we added in the transformation, then from the logic stated in the previous direction, the dominating set corresponds directly to the vertex cover. If some vertices come from the transformation, then we can substitute the vertex for either of the endpoints of the edge it corresponds to without changing the size of the dominating set. Thus, we can come up with a dominating set of size k that only uses vertices from our original problem, which will directly match to a minimum vertex cover of size k in the original problem.

Alternative Proof that Minimum Dominating Set is NP-Hard.
Reduction from Set Cover to Dominating Set
Minimum Set Cover is to find a set cover (a subset of all the sets) which is of size $\leq k$ where all elements are covered by at least one set of the set cover.

Suppose (S, U) is an instance of set cover where U denotes the set of all distinct elements and S is a set of subsets S_i of U. We will construct a graph $G = (V, E)$ as follows. For each element u in U construct a vertex; we will call these "element vertices". For each possible S_i construct a vertex; we will call these "set vertices". Connect each vertex S_i to all u in S_i.

Notice that if we were to run dominating set on the graph right now, we would be able to cover all the element vertices with any valid set cover, but we would have to pick every

single set vertex in order to ensure that all set vertices are covered. To rectify this, connect every set vertex to every other set vertex, forming a clique. This ensures that we can cover all the set vertices by picking just one. This way we really only need to worry about covering the element vertices.

Proof of Correctness of Reduction

1. **If minimum set cover has a solution, then minimum dominating set that corresponds to it has a solution.**

Suppose we have some minimum set cover of size k. This will correspond to a dominating set of size k as well. For each set in our minimum set cover, pick the corresponding set vertex. It follows directly from the construction of the graph and the definition of a set cover that all set and element vertices are covered.

2. **If minimum dominating set in the transformed format has a solution, then the corresponding minimum set cover has a solution.**

Suppose we have some dominating set D of size k. We can find a set cover of size $\leq k$. To do this, we will construct a new dominating set D' that contains only set vertices. Include every set vertex in D in D'. For each vertex in our dominating set that is an element vertex, pick any random neighboring set vertex and add it to D'. Observe that $|D'| \leq |D|$. Thus if there is a dominating set in G of size $\leq k$, there must be a set cover of size $\leq k$

Since this problem is in NP and is NP-Hard, it must be NP-Complete.

7 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, showing that 3-coloring is NP-hard.

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) 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?

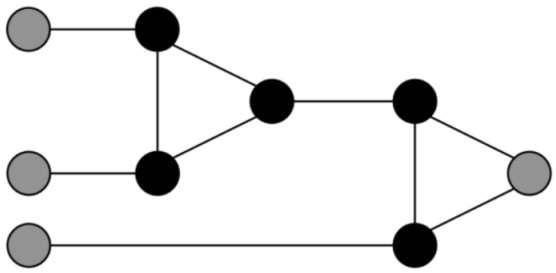
(b) Consider the following graph, which we will call a "gadget":

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.

(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) 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.

(b) 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.

(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.