COMPSCI 311: Introduction to Algorithms **Spring 2022**

Homework 6

Released 4/21/2022 Due 5/4/2022 11:59pm in Gradescope

Name: Brendan Cadogan Collaborators: Ben Tufano, Nare Courchesne, Anthony Ureña, Josh Shen
Instructions. You may work in groups, but you must write solutions yourself. List collaborators on your submission. You are allowed to have at most 4 collaborators. Also list any sources of help (including online sources) other than the textbook and course staff.

If you are asked to give an algorithm, please provide: (a) the pseudocode or precise description in words of the algorithm, (b) an explanation of the intuition for the algorithm, (c) a proof of correctness, (d) the running time of your algorithm and (e) justification for your running time analysis.

Submissions. Please submit a PDF file. You may submit a scanned handwritten document, but a typed submission is preferred. Please assign pages to questions in Gradescope.

1. **(30 points) SAT Variants** Show that each of the following restrictions of SAT are NP-complete:

a) Each clause contains three or fewer literals and each variable appears in three or fewer clauses.

First we need to show that this is NP. It is because we can use SAT to solve it because this is a specific restriction of SAT. Next, we need to prove that $3SAT \leq_p 1ASAT$ and that $1ASAT \leq_p 3SAT$. To do this, we'll find a polynomial-time reduction 3-SAT to 1a-SAT, and then another reduction 1a-SAT from 3-SAT. We can satisfy the reduction from 1a-SAT to 3-SAT by dealing with the three or fewer literals in each clause. First, if there are three literals in a clause, we do nothing. If there are less than three literals in a clause, we can do a few things, but the easiest is just adding a new variable that is always set to false so it doesn't affect the satisfiablity of the graph. Next we need to do the reduction 3-SAT to 1a-SAT. The only thing we need to do is meet the each variable appears in three or fewer clauses restriction. We can do this by iterating through the clauses, and when we find a variable in four or more clauses, we can substitute all but one instance of that variable with a new variable that is always equal to the value in the original variable. To make them always equal, we can add some clauses. So for example, if we have x appear in four clauses, we can change the xs in 3 of the clauses to y,z, and w, and add the following clauses, $(x, !y)$ and $(!x, z)$ and $(!z, w)$ and $(y, !w)$. Doing this creates circle logic so that x=y=z=w, and keeps all literals appearing in 3 clauses. Next we need to prove that this change doesn't change whether or not the formula is satisfiable. Let's say for some graph x, 3-SAT is satisfiable. Since our transformation is only substitutions that do not change the original value of the variable, 1a-SAT also satisfy x. The same logic applies when x can't be satisfied by 3-SAT. Now let's say for some graph z, 1a-SAT is satisfiable. This means that 3-SAT is also satisfiable because are change from 1a-SAT to 3-SAT is by adding false variables that don't change the value of the clauses because they are or clauses. The same applies when Z is not satisfiable. Now, since we know 1a-SAT is NP and NP-hard, it must also be NP-complete.

b) Each clause contains exactly three literals and each variable appears in four or fewer clauses.

First we need to show that this is NP. It is because we can use 3-SAT to solve it because this is a specific restriction of 3-SAT. Next, we need to prove that $3SAT \leq_p 1BSAT$ and that $1BSAT \leq_p 3SAT$. To do this, we'll find a polynomial-time reduction 3-SAT to 1b-SAT, and then a reduction 1b-SAT from 3-SAT. Luckily for us, since 1b-SAT contains exactly 3 literals, the reduction 1b-SAT from 3-SAT is already done and now the only thing we need to do is the four or fewer clauses part for the 3-SAT to 1b-SAT reduction. This can be easily done by iterating through the clauses, and finding all the variables that are in more than four clauses, and doing what we did in 1a, but we also need to add a false variable to meet the exactly three literals in each clause restriction. So for example, if we have x appear in five clauses, we can change the xs in 4 of the clauses to y,z,w, and m, and add the following clauses, $(x, !y, 0)$ and $(!x, z, 0)$ and $(!z, w, 0)$ and $(m, !w, 0)$ and $(!m, y, 0)$ where 0 represents a new false variable that is distinct for every clause. Doing so, we change each variable that appears in four or fewer clauses to appear now in three clauses, and we retain the logic that x=y=z=w=m. Next we need to prove that this change doesn't change whether or not the formula is satisfiable. Let's say for some graph x, 3-SAT is satisfiable. Since our transformation is only substitution, 1b-SAT is also satisfiable because if we replace an iterable with another iterable with the same value, it is still satisfiable. Next let's say for some graph y, 3-SAT is not satisfiable. We can use the same logic to determine that 1b-SAT is also not satisfiable. Now let's say there is some graph z that is satisfiable by 1b-SAT. We know it is also satisfiable by 3-SAT because 1b-SAT is an instance of 3-SAT. The same applies for when z is not satisfiable by 1b-SAT. Now since we know it is both NP and NP-hard, it is therefore NP-complete.

2. **(20 points) Quartered** Given natural numbers $a_1, a_2, \ldots, a_n$, is it possible to divide them into four sets with equal sums? Show that this problem is NP-complete.

First we need to show that quartered is NP. We know it is because we can verify it by looking at a solution, and checking that each sub division has the sum of the natural numbers divided by 4. This takes $O(n)$, which means the certifier is polynomial.

Next, we need to prove that $BinPacking \leq_p Quartered$ and $Quartered \leq_p BinPacking$. To do this, we need to do reductions from Quartered to Bin-Packing, and Bin-Packing to Quartered. First we will do Bin-Packing to Quartered. We can do this by limiting the amount of bins we have to four, and by changing the capacity of the bins to the sum of the natural numbers divided by 4. If the quotient is not a natural number, there is no answer for that instance of the problem. We then run Bin-Packing, and as long as all 4 bins are completely filled with no items left to pack, then there is a solution for quartered. Next we'll do from Quartered to Bin-Packing. To do this, the natural numbers represent the items, and the four subdivisions represent four bins with capacity of the natural numbers sum divided by four. Let's say for some instance x, Quartered is satisfiable. That means that bin-packing is also satisfiable, because quarters can be translated into a bin-packing with four bins of equal capacity. If some instance y of Quartered is unsatisfiable, that also means a 4 bin with equal capacity Bin-Packing is unsatisfiable for that instance because it means there is no way to sort the n items into 4 equal bins. Now let's say for some instance z, there is a 4 Bin-Packing solution that is satisfiable. That means that there is also a Quartered solution at that instance because Quartered is just Bin-Packing with 4 bins. The same logic can be applied to some instance w where 4 Bin-Packing is not satisfiable. This finishes the NP-hard proof, and since we know Quartered is also NP, it is therefore also NP-complete.

3. **(25 points) Rectangle Mosaic** You are given one large rectangle and n smaller rectangles, each with their integer dimensions. Your task is to completely cover the large rectangle with the small rectangles, without them overlapping. Show that this problem is NP-complete.

The first thing we need to do is prove that Rectangle Mosaic is NP. We can do this by finding a polynomial time certifier. The certifier checks that no rectangle overlaps, that all rectangles are within the borders of the bigger rectangle, that there is no empty space in the rectangle, and that the sum of the areas of the smaller rectangles is equal to the area of the bigger rectangle. This should be $O(n^2)$, which means that it is polynomial and therefore it is NP.

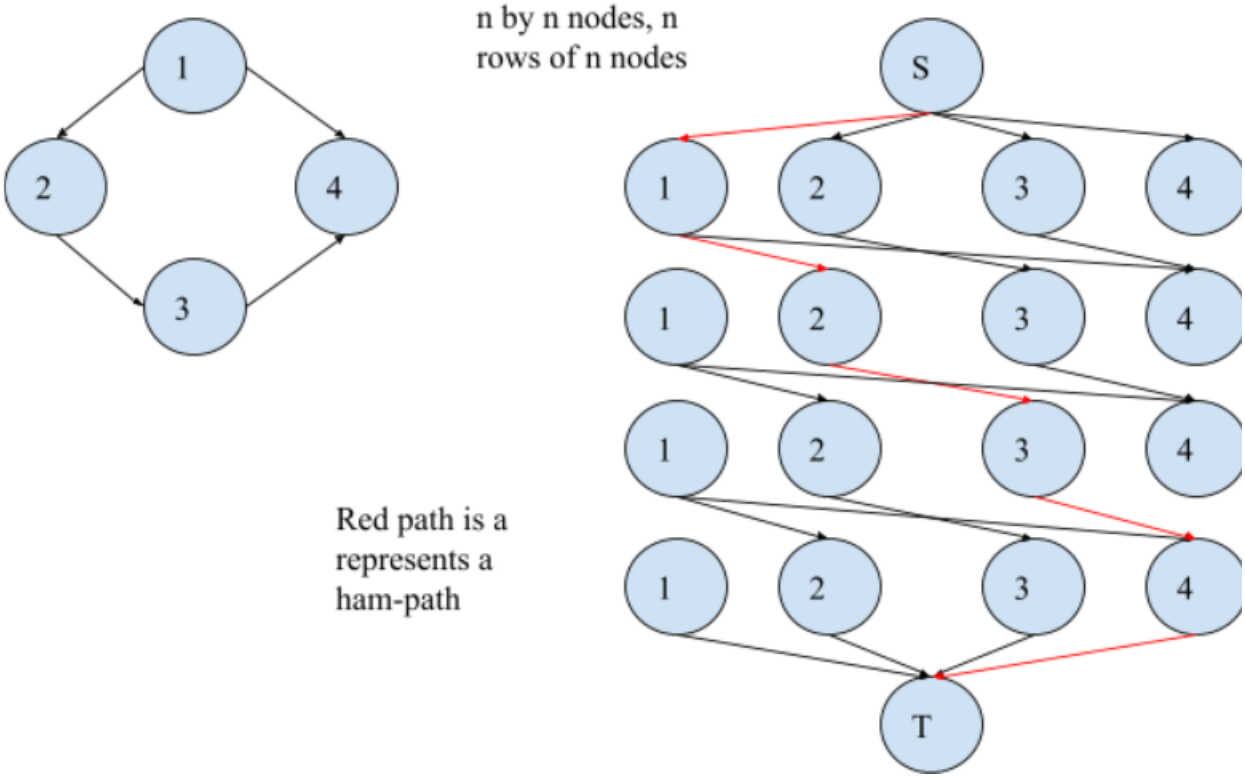
The next thing to do is to prove that Rectangle Mosaic is NP-hard, something we can do by proving that $RectangleMosaic \leq_p Partition$ and that $Partition \leq_p RectangleMosaic$. First we will use a specfic instance of Rectangle Mosaic in which the bigger rectangles, and all of the little rectangles have a height of 1. This can be reduced from Partition by saying the length of each rectangle is a number in partitions, and that the rectangle divided in half represents the two partitions in partitions. Now to go the other way, each number in partitions is a rectangle with a height of 1, and a width equal to the value of the number. Each bucket is a 1 by the sum of the bucket rectangle. Now, if for an instance x, we can solve partition, we can also solve Rectangle Mosaic when the big rectangle has a height of 1 because if there is a way to do two smaller rectangles, we can add them together to make one big rectangle. Now, if there is a satisfiable answer for Rectangle Mosaic for an instance y, then there may be a satisfiable answer for Partition because as long as you can divide the rectangle into two rectangles, then there is a satisfiable answer for Partition. This means that Rectangle Mosaic is NP-hard

Since we know that Rectangle Mosaic is both NP and NP-hard, it is therefore NP-complete.

4. **(25 points) Game of Guilds** You play an online game where you must get from the start to the goal choosing a path on a map. Around the countryside there are n inns, controlled by guilds that each use their own coins. Passing by an inn you must enter and pay one coin for each guild with a stake in the inn (inns and their guilds are marked on the map). You start out having exactly one coin for each of the p guilds in the country. Show that determining whether it's possible to choose a path and reach the goal is NP-complete.

First we need to prove that this problem is NP, which we can do by finding a polynomial time certifier. This is fairly trivial because we are given a path to the goal, so all we need to do is iterate over the path and make sure that we don't go to an inn when we don't have coins for its guild(s). This takes $O(n)$ which means the certifier is polynomial.

Next we need to prove that it is NP-hard. To do this, we can prove that $Ham - Path \leq_p GameofGuilds$ and that $GameofGuilds \leq_p Ham - Path$. First, we'll construct our guild graph. We'll start with a s and t node, and then n rows of n nodes. In this instance, each node represents 1 inn, and each inn has 1 guild, as well as each guild only having 1 inn. Next we draw 1 directed edge between s and all of the nodes in the first row, and then 1 directed edge between all of the nodes in the last row with t. Then to draw the directed edge between the rows, we draw an edge from one node to another node in the next row only if there is an edge from that node to the other node in the country map. Below is an example of this, on the left is the country map, and on the right is our constructed graph.

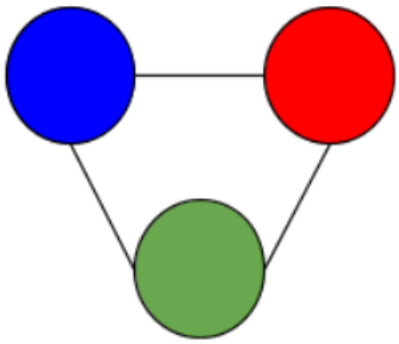


Our claim is that if there is a hamiltonian path in the country graph, then there is a path from s to t, and that means that there is a path to the goal. This is true because for our instances, we said that each inn has one guild, and each guild has one inn, so we won't run out of coins. And since there is a ham path, that means that no matter where the goal is, we can reach it because we can reach all nodes. Now, if there is a ham-path in the country graph, there is a path from s to t in the inn graph, because of the way we construct it. Furthermore, if there is a path from s to t, there must be a ham path because there are n by n nodes, and we can only go to a node once because we only have one coin, so that means we have gone to n distinct nodes and therefore a ham-path. This proves that GameofGuilds is NP-hard, and since we already proved that it is NP, it must be NP-complete.

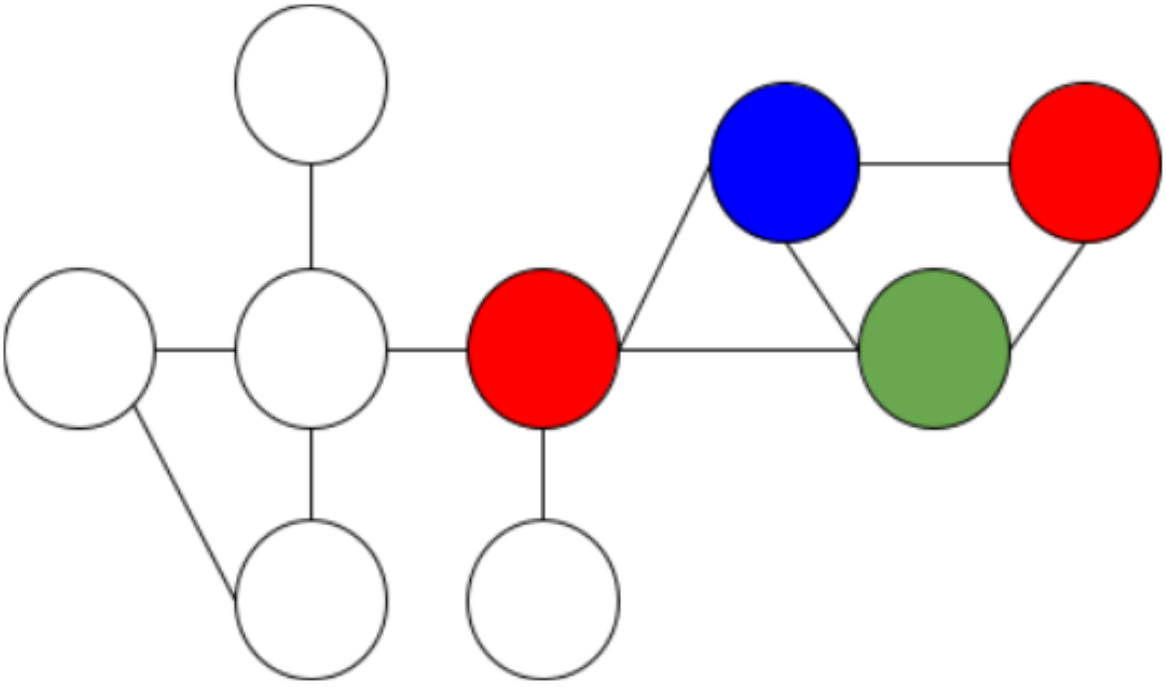
5. **(15 points) 3-Coloring** You have a procedure that given a graph answers whether it is 3-colorable. Using that procedure a polynomial number of times, find an actual 3-coloring of a graph, if it exists.

A. To do this, we will use two graphs, one being the original graph, and another being a graph of three nodes, each with an edge to each and each with one of the three colors.

This is the 3-color graph we use to verify node by node



Now we will draw two edges from the three node graph to our given graph from two of the nodes in our gadget. We run our procedure, and if it returns true, we know the color of that node is the color of the node it is not connected to. If it returns false, we try it again but with the other color substituted for one of the two we already checked. If this also returns false, we do it one more time to check the third color. If that also returns false, we know it is impossible. We continue this procedure for every node until we learn the color for every node.



B. Our intuition was from the lecture because we talked about gadgets in class.

C. The proof for this is very simple. If we force a node to be a certain color by using the gadget, and then run our procedure, we can figure out what color has to go there because our procedure will return true only if gadget with those edges is true, and false if the gadget with those edges is false.

D. The run time is $O(n)$.

E. The run time is $O(n)$ because we run the procedure at most 3n times which is $O(3nk)$ which we can say is $O(n)$.