Problem Set 5

Graded

Student

Anton Melnychuk

Total Points

60 / 61 pts

Question 1

Problem 0

1 / 1 pt

- ✔ **+ 0.2 pts** Name
- ✔ **+ 0.2 pts** SID
- ✔ **+ 0.2 pts** Collaborators and resources
- ✔ **+ 0.2 pts** Academic integrity policy
- ✔ **+ 0.2 pts** Hours spent

Question 2

Problem 1 - Maximizing Profit

20 / 20 pts

Algorithm and pseudocode

- ✔ **+ 3 pts** Correct vertex and edge construction (ignoring the direction of the edges).

  Note that there are two correct graphs: one where the order of layers is (s, J , M, t) and one where it is (s, M, J , t).

- ✔ **+ 1 pt** Correct direction of the edges (all edges should point from the source to the sink).

- ✔ **+ 1 pt** Computing the minimum cut

- ✔ **+ 3 pts** Correct relationship between minimum cut and maximum profit

Correctness

- ✔ **+ 2 pts** Creating a mapping from cuts to assignments (do not deduct any points if the students do not restrict this mapping to finite cuts)

- ✔ **+ 2 pts** Creating a mapping from assignments to cuts (do not deduct any points if the students do not restrict this mapping to finite cuts)

- ✔ **+ 1 pt** Claiming that any finite cut leads to a valid assignment

- ✔ **+ 1 pt** Proving the above statement

- ✔ **+ 2 pts** Proving that, for their mapping, any cut (A, B) is mapped to an assignment $(\mathcal{J}, \mathcal{M})$ (and vice-versa) such that $\pi(\mathcal{J}, \mathcal{M}) = \sum_i p_i - c(A, B)$

Running time

- ✔ **+ 2 pts** Correctly stated running time as O(|E| C), where C is an upper bound on the maximum flow

- ✔ **+ 1 pt** Explicitly or implicitly invoked that |E| ≤ mn

- ✔ **+ 1 pt** Explicitly or implicitly invoked that $C \leq \sum_i p_i + \sum_j q_j$

1. This is fine, but this edge case actually isn't necessary. In that case, your min cut would just be A={s} and B= {everything else}.

2. You need to show that this gives you a valid selection of jobs (all machines necessary are inlcuded, etc). I notice that you prove this in the next lemma, so not taking off points for this. But, in the future, you should include an explicit reference to lemma 2 if you're using it

Question 3

Problem 2 - Minimum-Value Partition

19 / 20 pts

✔ **+ 5 pts** Showed that Minimum Partition is in NP

Correct Construction of Reduction

✔ **+ 1 pt** Set $k = 2$

✔ **+ 1 pt** Set $T = \frac{1}{2}(\sum_i a_i)^2$

✔ **+ 3 pts** Set correct values of $a_1, a_2, \ldots, a_n$

Note it suffices to choose any $a_{n-1} = U + M$ and $a_n = \sum_i b_i - U + M$ where $M > (2\sqrt{2} - 1)\sum_i b_i$

+ 1 pt Claimed that the reduction can be constructed in polynomial time

---

(Completeness) If SS instance is YES then Min-partition instance is YES

✔ **+ 2 pts** Correct choice of S in terms of the set R

✔ **+ 2 pts** Prove that $V(S) \leq T$ if $\sum_{i \in R} b_i = U$

(Soundness) If Min-partition instance is YES then SS instance is YES

✔ **+ 2 pts** Claiming that, due to the value of T, a_{n-1} and a_n must be in different partitions in any partition S such that $V(S) \leq T$.

✔ **+ 2 pts** In any partition S where a_{n-1} and a_n belong to different partitions, $V(S) \leq T$ if and only if $\sum_{i \in S_1} b_i = U$

✔ **+ 1 pt** Combining the above two results to complete the proof

---

+ 0 pts Incorrect

Question 4

Problem 3 - From Knowing to Finding

20 / 20 pts

Algorithm

- ✔ **+ 3 pts** Correct algorithm for constructing G'
- ✔ **+ 3 pts** Correct algorithm to color G'

Correctness

- ✔ **+ 2 pts** Claiming and/or using the fact that G' has a valid 3 coloring
- ✔ **+ 3 pts** Any valid 3 coloring of G' is also a valid 3 coloring of G
- ✔ **+ 4 pts** Proving the correctness of the algorithm used to color G', e.g., by showing that two vertices (a, b) in G' have an edge between themselves if and only if they have the same color

Running time

- ✔ **+ 2.5 pts** Showed that at most $O(n^2)$ calls are made to the decision oracle.
- ✔ **+ 2.5 pts** Except for calling the oracle, the rest of the computation runs in $O(n^2)$ time

+ 0 pts Incorrect

Question assigned to the following page: 1

CPSC/ECON 365: Algorithms Yale University

Solution to Problem Set 5

Student Name: November 6, 2023 Due: Friday, November 17, 2023 at 5:00 pm ET

0 Your Information

(a) Your name.

Anton Melnychuk

(b) Your SID.

24787491

(c) A list your collaborators and any outside resources you consulted for this problem set.

ULAs: Ryan, Prastik, Christian, Anay

(d) Copy: "I have followed the academic integrity and collaboration policy as written above."

I have followed the academic integrity and collaboration policy as written above.

(e) How many hours did you spend in this problem set?

16 hours

No questions assigned to the following page.

1 Maximizing Profit

Suppose there are m different machines $M_1, \ldots, M_m$, and there are n different jobs $J_1, \ldots, J_n$. For each $1 \leq i \leq n$, to perform job J_i, we need to use a subset of the machines $R_i \subseteq \{M_1, \ldots, M_m\}$. Completing job J_i brings a **payment** of $p_i \in \mathbb{N}_0 = \{0, 1, 2, \ldots\}$ for each $1 \leq i \leq n$. Purchasing machine M_j **costs** $q_j \in \mathbb{N}_0$ for each $1 \leq j \leq m$.

If we purchase the machines $\mathcal{M} \subseteq \{M_1, \ldots, M_m\}$ and perform the jobs $\mathcal{J} \subseteq \{J_1, \ldots, J_n\}$, then our total payment is $\sum_{J_i \in \mathcal{J}} p_i$, our total cost is $\sum_{M_i \in \mathcal{M}} q_i$, and our **profit** is:

$$\pi(\mathcal{M}, \mathcal{J}) = \sum_{J_i \in \mathcal{J}} p_i - \sum_{M_j \in \mathcal{M}} q_j. \tag{1}$$

To ensure we can do the jobs, we need the following **feasibility** constraint:

$$\forall\, J_i \in \mathcal{J}, \qquad R_i \subseteq \mathcal{M}. \tag{2}$$

In the **Max-Profit** problem, you are asked to compute the maximum profit that you can get by purchasing a subset of machines and performing a subset of jobs. (Once we purchase a machine, we can use it any number of times. We can only perform each job once.) The input is $R = (R_1, \ldots, R_n)$ where each $R_i \subseteq \{M_1, \ldots, M_m\}$, $p = (p_1, \ldots, p_n) \in \mathbb{N}_0^n$, and $q = (q_1, \ldots, q_m) \in \mathbb{N}_0^m$.

Max-Profit(R, p, q)**:** Return the value of the maximum profit $\pi(\mathcal{M}, \mathcal{J})$ over all feasible choice $(\mathcal{M}, \mathcal{J})$ that satisfies (2).

Problem: Design an algorithm to solve **Max-Profit**:

1. Explain the reasoning of your algorithm and write it in pseudocode;
2. Prove the correctness of your algorithm; and
3. Analyze its running time.

To get full credit, your algorithm should run in time $O(mnT)$ where $T = \sum_{i=1}^{n} p_i + \sum_{j=1}^{m} q_j$.

(*Hint:* One approach is to reduce to **Min-Cut**, which we recall: The input is a network G, i.e. a directed graph (V, E) with edge capacities $c_e > 0$ for $e \in E$, a source $s \in V$ and a target $t \in V$.

Min-Cut(G)**:** Return the value of the minimum capacity $c(A, B)$ over any $s - t$ cut (A, B).

You must describe the reduction explicitly: define the vertices, edges, and capacities in the network G. Prove the correctness of your reduction: explain how the minimum capacity in G relates to the maximum profit in **Max-Profit**. Finally, use the reduction to write an algorithm for **Max-Profit**.)

Question assigned to the following page: 2

Solution. Transform the given problem into a network-flow framework. Create a directed graph consisting of two sets of vertices: one representing the machines $M_1, \ldots, M_m$ and the other representing the jobs $J_1, \ldots, J_n$. Edges are established between these sets if a machine M_k is part of a job J_i for any $1 \leq k \leq m$ and $1 \leq i \leq n$. These edges are assigned infinite capacity. Let's add a sink vertex t connected to all jobs with capacities equal to their respective payment $p_1, \ldots, p_n$ and a source vertex s linked to all machines with capacities of the costs $q_1, \ldots, q_m$. The network is designed such that all edges are directed from the source to the sink. Run the Ford-Fulkerson Algorithm to get the minimum cut. Then, the min-cut $c(A, B)$ problem yields an output of k (notation) with all the minimum edges going out from the machines/jobs out from the cut. We, in contrast, want to maximize the profit by taking all edges going from the jobs to have a maximum capacity. Then, one way to do this and receive the maximum profit is $\pi = \sum p_i - k$ (Proved Below), where k is the value of the cut $c(A, B)$. Finally, if the profit is negative, return 0 as every machine is so expensive that doing any job would produce loss, which is worse than doing no jobs. Same for the set of machines or jobs that are empty.

PseudoCode:

```
def max_profit(R, p, q):
    # check for base cases (in the solution)

    graph = initialize_graph(R, p, q) # f defined below
    min_cut_value = ford_fulkerson(graph, s, t)
    max_profit = sum(p) - min_cut_value
    profit = max(0, max_profit)

    return profit

def initialize_graph(R, p, q):
    graph = create_empty_graph()

    for i in range(len(R)):
        for machine in R[i]:
            connect_machine_to_job(graph, machine, i)

    connect_source_to_jobs(graph, p)
    connect_sink_to_machines(graph, q)

    return graph
```

Question assigned to the following page: 2

Lemma 1. Want to show that there always exist cut (including the minimum one), given the selection; and the opposite (for the sake of further proof).

To prove this, let's first consider two cases. Given a selection $\mathcal{J}, \mathcal{M}$, I will find the cut s.t. $\pi(\mathcal{J}, \mathcal{M}) = \sum p_i - v(c(A, B))$, where $A = \{\text{not } \mathcal{J}, \text{not } \mathcal{M}, t\}$ and $B = \{\mathcal{J}, \mathcal{M}, s\}$ and $A \cap B = \varnothing, s \in A, t \in B, A \cup B = V$. Similarly, the opposite. Given the the cut $c(A, B)$, I will find the selection $\mathcal{J}, \mathcal{M}$ such that $\pi(\mathcal{J}, \mathcal{M}) = \sum p_i - v(c(A, B))$, where $\mathcal{J} = \{\forall J_i \in B\}$ and $\mathcal{M} = \{\forall M_i \in B\}$. This means that we can go between these values (further used in the proof).

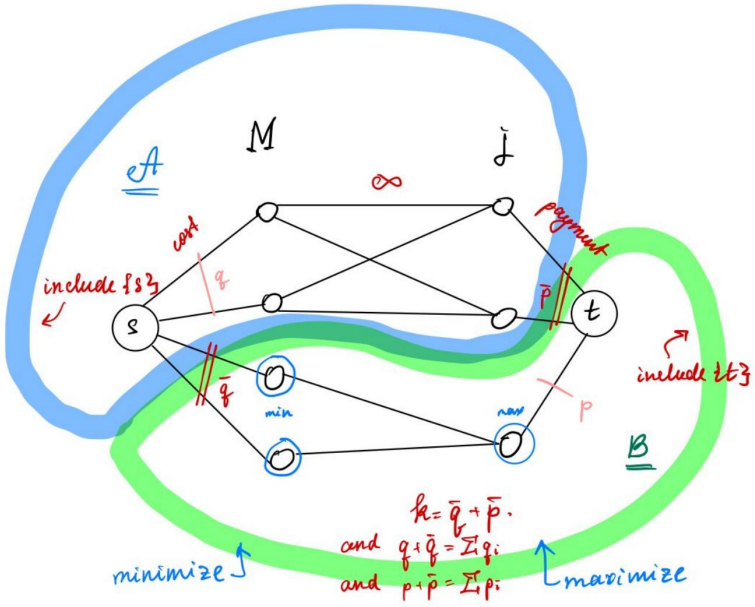

Figure 1: Visualization of the Problem with the MinCut k

Lemma 2. The output profit is not infinite, and for all jobs in any side of the cut there also exist all the machines necessary for its operation (validity of the mincut-selection).

Consider that the *mincut* divides the graph into two sets including source (A) and the sink (B). Suppose by contradiction that for some job there exist some machine that is on the opposite side B of the cut. Then, it means the cut crosses the edge in between set of J and M (Lemma 1), which

Question assigned to the following page: 2

by definition is set to the infinity. This leads to the contradiction as it cannot be the minimum cut, while there exist at least one way to cross the edges with not the infinite value. In particular, consider going through all edges connecting machines and the source s (by the definition). This also implies that regardless what valid selection of $\mathcal{M}$ and $\mathcal{J}$ we choose A or B of the cut, as long as it's true, there are no edges connecting jobs and machines such that the minimum cut be the infinite. Analogically, given a valid optimal selection, we can build a cut (Lemma 1) with the similar logic.

Proof. $\sum p_i - v(mincut(A, B))$ function output is equal to the profit π_{max}.

For simplicity let's denote the set of capacities of all edges going out from the cut $c(A, B)$ from the source s to any M_k as $\bar{q}$; let the set of all other edges to machines be q. Analogically, let's denote the set of all edges going to the sink from the jobs as the $\bar{p}$ and another set p. Now, given $mincut = \bar{q} + \bar{p}$, we aim to maximize the payment in $\sum p_i - \sum q_k$ by pursuing the set of jobs that are outside of the cut A (p), since the edges connecting them to the sink have not been crossed by the *mincut* during the executiong of the Ford-Fulkerson, so this set has a greater capacity. This selection of verticies (*mincut* side B) is valid from the Lemma 2 and Lemma 1, since we are working only with the vertices in the cut B, where the costs are minimized and the payments – maximized. Given that a new selection is valid, we can build the equation of desired algorithm optimal output (y) with $p - \bar{q}$ by considering some small rearrangements:

$$\pi_{max} = p - \bar{q} = p - (\bar{q} + \bar{p}) + \bar{p} = p - mincut + \bar{p} = \sum p_i - mincut$$

Since, in Lemma 1 we have proven that bidirectional relationship between *mincut*s and the valid optimal selection (by considering more general case), and in Lemma 2 we have shown other two important parts about the existence in the selection the necessary machines for any job to work and correct non-infinite output, and, finally, by direct manipulations of given *mincut* value (individually proofed during the class) and showing that the newly selected vertices are still in a valid selection (*mincut* B side) from Lemma 2, we know that the maximum profit is achieved with the equation depicted above. Please consider that the base cases are shown in the Solution. ■

Time Complexity. It requires $O(n + m)$ time to declare all vertices and the edges capacities. For each job, there may be at most n connections, so at most, building and checking the connections for each machine to any job, would take $O(nm)$ time ($mn+m+n$ edges). Finally, time complexity for finding the *mincut* based on the Ford-Fulkerson Algorithm is $O(mC)$, where the C is the capacities going out of the source (machine costs). Thus, the overall time complexity is $O(((\sum q_i)(mn+m+n))$ with some simplifications $O((\sum q_i)mn)$.

No questions assigned to the following page.

2 Minimum-Value Partition

Recall that a **partition** of $[n] \coloneqq \{1, \ldots, n\}$ is a collection of disjoint subsets $S = (S_1, \ldots, S_k)$ where $\emptyset \neq S_i \subseteq [n]$, $S_i \cap S_j = \emptyset$ for $i \neq j$, and $\bigcup_{i=1}^{k} S_i = [n]$. The **size** of a partition $S = (S_1, \ldots, S_k)$ is k, the number of subsets in S. Suppose each $i \in [n]$ has a *value* $a_i \in \mathbb{N}$. Given a partition $S = (S_1, \ldots, S_k)$ of $[n]$, we define the **value** of S as:

$$V(S) = \sum_{i=1}^{k} \left(\sum_{j \in S_i} a_j \right)^2 .$$

In the **Min-Partition** problem, you are asked to determine whether there is a partition of a certain size that has a small value. The input is $a = (a_1, a_2, \ldots, a_n) \in \mathbb{N}^n$, $k \geq 1$, and $T \in \mathbb{N}$.

> **Min-Partition**(a, k, T)**:** Return `Yes` if there is a partition $S = (S_1, \ldots, S_k)$ of $[n]$ such that $V(S; a) \leq T$; else, return `No`.

Problem: Prove that **Min-Partition** is NP-complete. That is, prove that (1) **Min-Partition** is in NP, and (2) **Min-Partition** is NP-hard.

(*Hint:* One approach is to reduce from **Subset-Sum**, which we recall: The input is $b = (b_1, \ldots, b_n) \in \mathbb{N}^n$ and $U \in \mathbb{N}$. You may assume **Subset-Sum** is NP-hard.

> **Subset-Sum**(b, U)**:** Return `Yes` if there is $R \subseteq [n]$ such that $\sum_{i \in R} b_i = U$; else, return `No`.

You may also find the following fact useful: for each $y \in \mathbb{R}$, the function $f(x) = (x - y)^2 + x^2$ has a unique minimum at the point $x = y/2$.)

Question assigned to the following page: 3

Solution. To prove **NP-Completeness**, consider two parts of the problem:

(a) Min-Partition is in NP.

The certifier checks whether a given partition $S = (S_1, \ldots, S_k)$ of $[n]$ satisfies the Min-Partition property $V(S; a) \leq T$ (returns True) for any arbitrary S and a provided for verification. To do this, we need directly plug given certificate in the value function $V(S)$ and check if all subsets satisfy $\emptyset \neq S_i \subseteq [n]$, $S_i \cap S_j = \emptyset$ for $i \neq j$, and $\bigcup_{i=1}^{k} S_i = [n]$ properties. In the function, we utilize two nested loops and before entering the outer one, we square all the values in constant time $O(1)$ (please, consider the definition of the value of S). In the inner loop, we iterate through all possible values in S_i, which is at most n, and in the outer loop, we consider the worst-case scenario of the partition in all sets with one element, taking n time as well. Although, the time complexity might be lower, we still see that the certifier takes at most $O(n^2)$, which is polynomial. Finally, checking all the other properties would require polynomial time e.g. we sum up all subsets (including duplicates) and check if it is equal to $[n]$.

(b) Min-Partition is NP hard for the instance (k=2).

To establish the reduction (algorithm), we consider a specific partition of $[n]$ for $k = 2$. Let S be this partition, and we denote its subsets as S_1 and S_2. Specifically, we have $S = (S_1, S_2)$. To get the value of T, consider some rearrangements:

$$V(S) = (\sum_{x \in s_1} x)^2 + (\sum_{y \in s_2} y)^2$$

$$= (\sum_{x \in s_1} x)^2 + (\sum_{\forall a_i \in a} a_i - \sum_{x \in s_1} x)^2$$

$$= (\sum_{x \in s_1} x)^2 + (\sum_{x \in s_1} x - \sum_{\forall a_i \in a} a_i)^2$$

$$\implies \sum_{x \in s_1} x = (\frac{1}{2} \sum_{\forall a_i \in a} a_i) \text{ (from the hint)}$$

Since, we expect them being equal, for $k = 2$ we expect T be twice larger ($k = 2$):

$$T = 2(\frac{1}{2} \sum_{\forall a_i \in a} a_i)^2$$

This selection ensures that T represents the minimal achievable value for the Min-Partition problem when the partition has two subsets.

Question assigned to the following page: 3

Now, given the value of T let's reduce the Subset-Sum to Min-Partition, by transforming it more to the similar problem of the Subset-Sum with $k = 2$ arrays.

$$\text{Let, } B = \sum b_i \text{ for the further proof}$$

.

One of the ways to apply reduction from the Subset-Sum:

$$B + (B + U) + (2B - U) = 4B, \text{ where } 4B \text{ is sum of all elements in } a$$

Let's show the corectness by constructing the byjection. Let's denote given inputs to the algo:

$$A = \{b_1, \ldots, b_n, (B + U), (2B - U)\}$$

$$\text{sum of elements in } S_1 = S_2 \ (= 2B \text{ expected})$$

$$T = 2(\frac{1}{2} \sum_{\forall a_i \in a} a_i)^2$$

$\Longleftarrow$ **Assume there exists a Min-Partition** $S = (S_1, \ldots, S_k)$ **s.t.** $V(S; a) \leq T$**. [WTS: there exists a Subset-Sum solution** $R \subseteq [n]$ **s.t.** $\sum_{i \in R} b_i = U$**].**

Consider two cases. $(B + U)$ and $(2B + U)$ last two elements in a are in the same subset; or these elements are in the different ones.

Case 1: Same Subset. In this scenario, let's evaluate the sum of these two elements: $(B + U) + (2B - U) = 3B$. However, since $V(S) = (9B^2 + ...) > 8B^2 = T$ (from the definition of T), this contradicts the assumption that $V(S; a) \leq T$. ■

Case 2. Different Subset. Then, consider two such partitions:

$$s_1 = \{b_1, \ldots, b_j, (B + U)\}$$

$$s_2 = A \setminus s_1$$

where $\{b_1, \ldots b_j\} \in b$

We aim to verify if there exists at least one instance of a partition such that, given the Subset-Sum input U, we can achieve the corresponding Min-Partition:

$$2B = X + (2B - U)$$

where X is the sum of all elements in s_2 besides $(2B - U)$

Question assigned to the following page: 3

$$2B = X + (2B - U)$$

$X = U$, what we wanted to prove. ■

$\implies$ **Assume there exists a subset $R \subseteq [n]$ such that $\sum_{i \in R} b_i = U$. [Need to show that there exists a partition $S = (S_1, \ldots, S_k)$ such that $V(S; a) \leq T$].**

Constructing S_1, S_2:

$$s_1 = Y \cup \{2B - U\} \text{ and}$$

$$s_2 = A \setminus s_1, \text{ where } Y = \{b_i \mid i \in R\}$$

Then,

$$\sum_{s_1} a = U + (2B - U) = 2B$$

$$\sum_{s_2} a = (B - U) + (B + U) = 2B$$

$$V(S) = (\sum_{x \in s_1} x)^2 + (\sum_{y \in s_2} y)^2$$

$$= (2B)^2 + (2B)^2 = 8(B)^2$$

Finally,

$$T = 2(\frac{1}{2} \sum_{\forall a_i \in a} a_i)^2 = 2(\frac{1}{2} 4B)^2 = 8(B)^2$$

Which are the same values. Therefore $V(S) \leq T$, what was needed to prove. ■

Now, we can establish that Min-Partition is NP-hard, as we have reduced Subset-Sum to it. Consequently, we have demonstrated both parts, affirming that Min-Partition is **NP-Complete**.

No questions assigned to the following page.

3 From Knowing to Finding

In class we usually discuss the *decision* version of problems, which have `Yes` or `No` answers (for example, whether a graph has a 3-coloring). Often, we also want to find a witness that certifies the answer (for example, find a 3-coloring in the graph). In many cases, we can find a polynomial-time reduction from the *search* problem to the *decision* problem. Here we consider the 3**-Coloring** problem.

Recall a 3**-coloring** of a graph $G = (V, E)$ is a map $c\colon V \to \{1, 2, 3\}$ such that $c(u) \neq c(v)$ for all $(u, v) \in E$. Consider the *decision* and *search* versions of the **3-Coloring** problem:

Dec-3-Coloring(G): Return `Yes` if G has a 3-coloring; else, return `No`.

Search-3-Coloring(G): Return a 3-coloring in G if one exists; else, return `No`.

Problem: Provide a polynomial-time reduction from **Search-3-Coloring** to **Dec-3-Coloring**.

(*Hint:* Note carefully the input of **Dec-3-Coloring** is a graph. An algorithm **A** for **Dec-3-Coloring** takes in a graph and outputs a `Yes` or `No` answer. Think about what property makes a graph easy to 3-color, then try to transform your graph to have that property; the algorithm **A** will be useful to find that transformation. Explain your reasoning, and prove the correctness of your reduction.)

Question assigned to the following page: 4

Solution. **3-Coloring Reduction:**

```
First, check if the given graph G is 3-colourable.
If not -> Return No (No Valid Coloring).

For each vertex in the given graph:
    Try iteratively adding edges between all other nodes
    not including the current vertex, checking with
    the 3-Decision problem if there is a way to build
    such a new graph G'. If it is possible,
    keep the edge; if not -> skip it.

1. Given a new graph G', choose (randomly) a source vertex (s)
and color it blue in G (or any other color not taken yet). Color
the rest vertices that are not connected to the current vertex
by an edge in G' using the same color (in this case, blue).
One way to do this is to run BFS/DFS on G'.

2. For any first neighbor n of s not-colored in G':
    While there exist one (not-colored neighbor):
        i. Color n with a new/different color in G (e.g., green).
        ii. Set all other vertices that are not connected directly
            to n in G' to the same color (run through all by BFS).

Return the graph G with a valid coloring.
```

Reduction Time Justification. Building a complementary graph would require creating at most $|V|(|V|-1)/2$ edges between all unconnected vertices with $O(|V|^2)$ time preprocessing. Then, coloring the graph would require going though each edge and node and coloring them in necessary color, so for 3 colors it does $O(3 * (|V| + |E|))$. Thus, overall complexity of the reduction can be described in the polynomial time reduction $O(|V|^2(|V| + |E|))$, which is polynomial. In particular it would be at most $O(|V|^3 + |E||V|^2)$. This is a polynomial time complexity for the reduction.

Question assigned to the following page: 4

Proof.

Fact. If there exists a 3-Colorable graph G' s.t. there exist all edges between all pairs of vertecies that do not have the same color, then this graph G' is colorable.

Consider by contradiction that this graph is not colorable. Then by definition of the problem, there exist at least one edge that connects both vertices that have the same color, but this leads to the contradiction since the Dec-3-Coloring problem would reject this connection by the setup.

Lemma 1. Given a colorable graph G, I can constuct graph G' that follows the fact property.

During the execution of the algorithm, we consider all possible combinations of the edges between two vertices in any order of colors. Consider by the contradiction that there, in contrast, exist and edge that does not connect different colors or there exist extra edge connecting vertices with the same color. That means that at some iteration of the problem, the algorithm adds edge between 2 same colors and 3-Dec-Problem returns Yes, which could not happen by the assumption. Otherwise, if the final graph G' does not have edge between some vertices, that would say the opposite $\implies$ algorithm considered 2 vertices that are different, but returned False. Both cases leads to the contradiction. Therefore, there must be at least one coloring in the G'.

Lemma 2. Given colored complementary graph G', there exist valid coloring in the graph G such that each component built by choosing arbitrary node s and its uncolored neighbors (in particular component type of $\{s\} \cup \{\forall v \in V.(s, v) \notin E\}$) has its own color.

Since, the graph G' has been build from the colorable graph $G \implies G \subseteq G'$ (is the subset). Therefore, all edges in the graph G are also in the graph G'. By coloring graph G by ignoring (say deleting) edges that are not in the G, but in G', we delete ones that have different color, therefore, if there exist coloring of vertices in the G', removing edge to G would not affect coloring, since they still can be different colors. By repetition of this step, we reach G and prove that it's colorable ■.