CPSC 320 2021W2: Assignment 5

This assignment is due **Monday December 5, 2022 at 22:00 Pacific Time**. Assignments submitted within 24 hours after the deadline will be accepted, but a penalty of up to 15% will be applied. Please follow these guidelines:

- Prepare your solution using LaTeX, and submit a pdf file. Easiest will be to use the .tex file provided. For questions where you need to select a circle, you can simply change `\fillinMCmath` to `\fillinMCmathsoln` .
- Enclose each paragraph of your solution with `\begin{soln}Your solution here...\end{soln}`. Your solution will then appear in dark blue, making it a lot easier for TAs to find what you wrote.
- Start each problem on a new page, using the same numbering and ordering as this assignment handout.
- Submit the assignment via GradeScope at `https://gradescope.ca`. Your group must make a **single** submission via one group member's account, marking all other group members in that submission **using GradeScope's interface**.
- After uploading to Gradescope, link each question with the page of your pdf containing your solution. There are instructions for doing this on the CPSC 121 website, see `https://www.students.cs.ubc.ca/~cs-121/2020W2/index.php?page=assignments&menu=1&submenu=3`. Ignore the statement about group size that is on that page.

Before we begin, a few notes on pseudocode throughout CPSC 320: Your pseudocode should communicate your algorithm clearly, concisely, correctly, and without irrelevant detail. Reasonable use of plain English is fine in such pseudocode. You should envision your audience as a capable CPSC 320 student unfamiliar with the problem you are solving. If you choose to use actual code, note that you may **neither** include what we consider to be irrelevant detail **nor** assume that we understand the particular language you chose. (So, for example, do not write `#include <iostream>` at the start of your pseudocode, and avoid language-specific notation like C/C++/Java's ternary (question-mark-colon) operator.)

Remember also to **justify/explain your answers**. We understand that gauging how much justification to provide can be tricky. Inevitably, judgment is applied by both student and grader as to how much is enough, or too much, and it's frustrating for all concerned when judgments don't align. Justifications/explanations need not be long or formal, but should be clear and specific (referring back to lines of pseudocode, for example). Proofs should be a bit more formal.

On the plus side, if you choose an incorrect answer when selecting an option but your reasoning shows partial understanding, you might get more marks than if no justification is provided. And the effort you expend in writing down the justification will hopefully help you gain deeper understanding and may well help you converge to the right selection :).

Ok, time to get started...

1 Statement on Collaboration and Use of Resources

To develop good practices in doing homeworks, citing resources and acknowledging input from others, please complete the following. This question is worth 2 marks.

1. All group members have read and followed the guidelines for groupwork on assignments in CPSC 320 (see `https://www.students.cs.ubc.ca/~cs-320/2022W1/index.php?page=assignments&menu=1&submenu=3`).

   [✓ Yes] No

2. We used the following resources (list books, online sources, etc. that you consulted):

3. One or more of us consulted with course staff during office hours.

   Yes [✓ No]

4. Either none of us collaborated with other CPSC 320 students; or, one or more of us collaborated with other CPSC 320 students but none of us took written notes during our consultations and we took at least a half-hour break afterwards.

   Yes [✓ No]

   If you collaborated with other CPSC 320 students, please list their name(s) here:

5. Either none of us consulted with others outside of CPSC 320; or, one or more of us consulted others outside of CPSC 320 but none of us took written notes during our consultations and we took at least a half-hour break afterwards.

   Yes [✓ No]

   If you consulted others outside of CPSC 320, please list their name(s) here:

2 Dynamic Programming Vs. Zombies

The kingdom of Svenyarnia is about to wage battle against the Zombie King. The Zombie King leads an army of the undead, and he can raise all slain soldiers from the dead to join his ranks. This ability obviously gives him a huge military advantage, but the Svenyarnian strategists want to know just how big this advantage is. In particular, they want to know how big an army they would need to muster in order to have a reasonable chance of defeating the zombies.

The battle proceeds as follows. The soldiers from each army line up in single file. One soldier comes forward from each line and the pair will duel. If the living soldier wins, the zombie is (permanently) killed and the soldier moves to the back of his own line. If the zombie wins, the soldier dies and is instantly resurrected as a zombie and goes to the back of the zombie army's line, along with the victorious zombie. The battle continues until one army is eliminated. At each duel, there is a (constant) probability α between 0 and 1 that the living soldier will defeat the zombie soldier.

1. **[3 marks]** Give a recurrence to define $P(m, n)$, which describes the probability that a living army of size m will defeat a zombie army of size n.

$$P(m,n) = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{if } m = 0 \\ \alpha * P(m, n-1) + (1-\alpha) * P(m-1, n+1) & \text{if } m > 0 \wedge n > 0 \end{cases} \tag{1}$$

2. **[5 marks]** Write pseudocode for an iterative dynamic programming algorithm to compute $P(m, n)$.

```
procedure P($M$, $N$)
    let $cache$ be an empty dictionary
    for $i$ in range(1, $N + M$) do                                   ▷ $(1, N + M)$ inclusive
                        ▷ Every combination of $M$ being 0 (no more soldiers) has a probability of 0
        $cache[“0” + “-” + i.toString()] = 0$

                        ▷ Every combination of $N$ being 0 (no more zombies) has a probability of 1
        $cache[i.toString() + “-” + “0”] = 1$
    end for
    for $i$ in range(1, $M$) do                                           ▷ $(1, M)$ inclusive
        for $j$ in range(1, $N + (M - i)$) do                         ▷ $(1, N + M - i)$ inclusive
                                    ▷ Current combination of $i$, $j$ that we will compute
            let $current = i.toString + “-” + j.toString()$

                          ▷ If the soldier in front wins, this is the next case: $P(i, j - 1)$
            let $win = i.toString + “-” + (j - 1).toString()$

                      ▷ If the soldier in front loses, this is the next case: $P(i - 1, j + 1)$
            let $loss = (i - 1).toString + “-” + (j + 1).toString()$

                      ▷ The soldier has $\alpha$ chance of winning and $1 - \alpha$ chance of losing
            $cache[current] = \alpha \cdot cache[win] + (1 - \alpha) \cdot cache[loss]$
        end for
    end for
                    ▷ All calculations are done, we now return the answer for our case $M$, $N$
    return $cache(M.toString() + “-” + N.toString())$
end procedure
```

3. **[3 marks]** Give and briefly justify an asymptotic bound on the runtime and memory use of your algorithm for 2.2, in terms of m and n.

Run-time Analysis:

The run-time is determined by the number of times the loops iterate. The nested for-loops will be more expensive than the first for-loop, so we will study the nested loops for the asymptotic bound on run-time.

If we had a case where $M = 5$ and $N = 4$, then the following case if the first soldier loses leads to $M = 4$ and $N = 5$. If the next soldier loses also, we get the case $M = 3$ and $N = 6$, and so on: $M = 2$ and $N = 7$, $M = 1$ and $N = 8$, $M = 0$ and $N = 9$.

This means the inner loop will be longest for smaller values of i (representing M) and lessen incrementally until we get to $i = M$. To compute this, we made the inner loop iterate from 1 to $N + M - i$, where i would be defined by the outer loop (iterating M times).

Hence, the outer loop iterates M times and the inner loop iterates, in the longest case, approximately, $M + N$ times.

We get $M \cdot (M + N)$ iterations, each taking constant time, giving $M^2 + MN$.

We don't know which one is bigger, M^2 or MN, so our asymptotic bound on run-time is $O(max(M^2, MN))$.

Memory Analysis:

The space complexity is determined by how much data we store in the *cache* dictionary.

Let's see how much space each key-value pair in the dictionary will take. The keys, based on our design, are of type string. It simply takes i and j, converts them to strings, and joins them with a dash between them. The value of each pair is a numeric/decimal value indicating the probability that the army wins with the number of soldiers i and zombies j, defined by the key. Overall, each key-value pair takes constant space, storing a string for the key and a number for the value.

Therefore, the number of key-value pairs our *cache* dictionary will store is the space complexity of the algorithm, since each key-value pair takes constant space.

Again, we will study the nested loops mainly since they run more times than the first for-loop.

Our dictionary stores one key-value pair on every iteration of the inner loop. We already determined in the run-time analysis that this runs $M^2 + MN$ times.

Thus, we will store $M^2 + MN$. key-value pairs in our dictionary too, leading to a space complexity of $O(max(M^2, MN))$.

3 Clustering 2.0/Colour Me Puzzled 3.0

Recall the photo categorization problem described in worksheets 5 and 6. An instance of the problem is given by

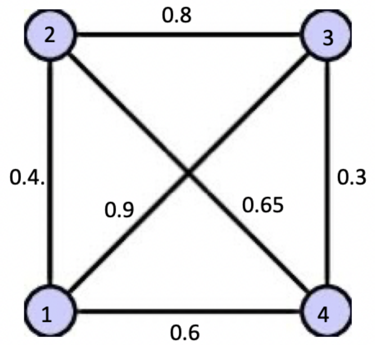
- n, the number of photos (numbered from 1 to n);
- E, a set of weighted edges, one for each pair of photos, where the weight is a similarity in the range between 0 and 1 (the higher the weight, the more similar the photos); and
- c, the desired the number of categories, where $1 \leq c \leq n$.

A solution to this problem is a *categorization*, which we define as a partition of photos into c (non-empty) sets, or categories. In the version of this problem we dealt with in class, our goal was to find the categorization that minimized the maximum inter-category edge weight. Suppose now that we change the goal of the photo categorization to **maximize the minimum intra-category edge similarity**. We call this the *Max-Min Clustering Problem* (MMCP).

1. **[3 marks]** In class, we saw that an optimal greedy algorithm for minimizing the maximum inter-category edge weight was to initialize all photos in their own category and merge the photos adjacent to the highest-weight intercategory edge until we achieved the desired number of categories. Below is a greedy algorithm for MMCP, which starts with all photos in a single category and separates the photos adjacent to the lowest-weight intracategory edge until we achieve the desired number of categories.

```
function MMCP-Greedy($n, E, c$)
▷ $n \geq 1$ is the number of photos
▷ $E$ is a set of edges of the form $(p, p', s)$, where $s$ is the similarity of $p$ and $p'$
▷ $c$ is the number of categories, $1 \leq c \leq n$
    create a list of the edges of $E$, in increasing order by similarity
    let $\mathcal{C}$ be the categorization with all photos in one category
    Num-$\mathcal{C} \leftarrow 1$            ▷ Initial number of categories
    while Num-$\mathcal{C} < c$ do
        remove the lowest-similarity edge $(p, p', s)$ from the list
        if $p$ and $p'$ are in the same category $S$ of $\mathcal{C}$ then
            ▷ split $S$ into two new categories $T$ and $T'$ as follows:
            put $p$ in $T$
            put $p'$ in $T'$
            for each remaining $p''$ in $S$ do
                put $p''$ in $T$ if $p''$ is more similar to $p$ than $p'$
                put $p''$ in $T'$ otherwise
            end for
            ▷ now $S$ is replaced by $T$ and $T'$ in $\mathcal{C}$
            Num-$\mathcal{C} \leftarrow$ Num-$\mathcal{C} + 1$
        end if
    end while
    return $\mathcal{C}$
end function
```

Give and explain a 4-node counterexample to show that this greedy algorithm is not optimal (i.e., does not always produce a solution that maximizes the minimum intra-category edge weight).



With the 4-node graph above, the greedy algorithm will not be optimal. First, the algorithm will group every photo in 1 category S, then it starts by removing the lowest-similarity edge which is (3,4,0.3). Since $p3, p4$ are in the same S, they are split into T, T' respectively. Then, the remaining photos are $p1, p2$ which are both more similar to $p3$ than $p4$, thus $p1, p2, p3$ will be in the same category T, and $p4$ will be in another category T'. Now, that the number of categories reach 2, the loop terminates. The cluster $p1, p2, p3$ has a minimum weighted edge of (1,2,0.4). However, this does not maximize the minimum intra-category weight, because if we group (p1,p3), (p2,p4), we can achieve a higher minimum intra-category weight of 0.65. Thus, the greedy algorithm is not optimal.

2. **[3 marks]** We've stated MMCP as an optimization problem. Give the decision version of MMCP, and prove that MMCP is in NP.

Add an extra parameter r that is an integer and $0 \leq r \leq 1$ and ask whether there is a partition of n photos into c clusters such that no weighted edge in the same cluster have a weight less than r.
The decision version is in NP because given the certificate which is the c clusters of photos (C1,C2,...,Cc), we can verify in polynomial time if the cluster is a complete graph and if any intra-edge in the cluster have a weight less than r. ("YES" if no such edge exists, "NO" if any intra-edge $<$ r)
First, we can verify that the number of clusters is less than or equal to c through a simple loop. Then, for each clusters C_k where $1 \leq k \leq c$, we can verify that $w(p_i, p_j) \geq r$ for all pair $p_i, p_j \in C_k$ and $i \neq j$. We can assume that a cluster with 1 photo would have a $w(p,p) = 1$. In the worst case, we have to go through every edges once, and since there are $n(n-1)/2$ possible edges in a complete graph, then the algorithm is bounded by O(n^2).

3. **[4 marks]** We will now show that MMCP is NP-hard (which, together with the proof from 3.2 that MMCP is in NP, means that it is NP-complete). We do this by reducing a known NP-hard problem to it. For this problem, you must do a reduction from the graph colouring problem, which (as a decision problem) is: given a graph G and a bound k, can we colour the vertices of G using no more than k colours such that no two adjacent vertices share a colour? You may assume that $k \leq n$, where n is the number of vertices in G.

   For this part of the question, you do not need to prove the correctness of your reduction (that comes next), but you should explain the reasoning behind your reduction and why it runs in polynomial time.

   Given a graph G=(V,E) and a bound k, we can reduce the instance to MMCP as follow:
   Label all vertices $v \in V$ as p_v and let $w(p_i, p_j)$ be the weight between all vertices i, j such that:

$$w(p_i, p_j) = \begin{cases} 1 & \text{if } (i,j) \notin E \land (i \neq j) \\ 0 & \text{if } (i,j) \in E \land (i \neq j) \end{cases} \tag{2}$$

   Let $c = k$, to check if it is possible to partition the vertices into k clusters (1 cluster = 1 color).
   This reduction can be done in polynomial time because given a graph G=(V,E), we only need to build a complete graph of |V| vertices and |V|*(|V|-1)/2 edges and label each edges a weight of either 0 or 1. Thus, the reduction runtime is O($|V| * (|V| - 1)/2$) or equivalently O(n^2), where n is the number of vertices in G.
   Logically, we will partition a unique color to each unique cluster ($C1, .., Cc$) (each vertices in the same cluster can be the same color, if the certifier returns "YES"). If any pair of vertices ($\in$ V) that are connected by an $e \in E$, then they cannot be the same color, so we label their weight as 0 (minimum possible similarity) meaning that they would not return a "YES" answer if that pair of vertices are in the same cluster. Otherwise, if any vertices are not connected, then they can be the same color, so we label their weight as 1 (maximum possible similarity) meaning that they could be in the same cluster and still returns a "YES".

4. **[3 marks]** To prove your reduction correct, you must prove that the answer to the MMCP instance is YES if and only if the answer the answer to the graph colouring instance is YES. For this part of the question, prove that if the answer to the graph colouring instance is YES, then the answer to your reduced MMCP instance is YES.

   If the answer to GC instance is YES, that would mean there is a way to color all adjacent vertices differently using only k colors.
   Claim: ONLY vertices with the same colors can be partitioned in the same clusters.
   Proof: The reduction from GC to MMCP would have labelled any adjacent pairs of vertices as $w(p_i, p_j) = 0$ and any non-adjacent pairs of vertices as $w(p_i, p_j) = 1$. In GC, any pair of vertices with the same colors cannot be adjacent (that would violate GC restrictions), so ONLY pairs of vertices v_i, v_j such that $(i \neq j) \land (v_i, v_j \in V) \land ((i,j) \notin E)$ can be partitioned into the same clusters as that would maximize the weight/similarity to 1 (otherwise a weight of 0 would be intra-category thus not maximized). Suppose otherwise, an originally adjacent vertices are in the same cluster, then that would mean that the minimum intra-weight of that cluster would be 0 which would have resulted in a NO. Thus, if a GC instance is YES, then, with only k colors, each of the adjacent vertices can be colored differently, so it can be reduced to MMCP such that ONLY non-adjacent vertices with the same colors will be partitioned in the same clusters resulting in a YES (because non-adjacent vertices will be labelled a weight of $1 \geq r$)

5. **[3 marks]** Now for the other direction of the if-and-only-if: prove that if the answer to the reduced MMCP instance is YES, then the answer to the original graph colouring instance is also YES.

Because the answer to reduced MMCP instance is YES, there exists c clusters of vertices/photos with all edges within the cluster (intra-weight) greater equal to r. The clusters produced after the reduction are complete graphs, each cluster represents a unique color for the vertices within.
Claim: for each originally non-adjacent pair of vertices v_i, v_j such that $(i \neq j) \wedge (v_i, v_j \in V) \wedge ((i, j) \notin E)$ in GC instance, they can either be part of the same or different cluster in MMCP solution and still produces a YES (meaning that they can be colored the same or different).
Proof:
First, if the pair of vertices are in different clusters, then it would not matter because we do not care about inter-category weight.
Suppose by contrapositive: Assume the answer to original GC is not YES by letting v_i, v_j be originally adjacent pair of vertices in GC and ended up in the same cluster. This would mean that their weight would be labelled 0 during the reduction, so the solution would not be YES, because the intra-weight is minimized to 0. Conclusion: if the original GC instance is not YES, then the reduced MMCP is not YES. Thus, if the reduced MMCP is YES, then original GC is also YES, by contrapositive and because all clusters include only originally non-adjacent vertices (their weights $= 1 \geq r$ after reduction).
conversion to GC: choose a different color for each unique clusters, then color each vertices within the same clusters the same color. To achieve the original GC instance: build a complete graph (connecting all the clusters) then remove the edges for each pair of vertices that were in the same clusters. Now, each adjacent vertices will be different colors (because they were in different clusters), thus that GC instance is also YES.

4 Greater Than or Equal to Three's a Crowd

In the tutorial quiz, we considered the following SMP-like problem. You're in charge of matching up n employers and n computer science co-op students for summer internships. Instead of having ranked preference lists, each employer has a list of students they're willing to work with and vice versa. You want to know if it's possible to generate a perfect matching in which everyone is paired with someone they're willing to work with. On the tutorial quiz, we saw how to solve this problem with a polynomial-time reduction to the Bipartite Matching Problem (described on the tutorial quiz and in Section 7.5 of the textbook). We call this the *CO-OP2 problem*.

We now consider the case where each employer now wants to hire a computer science student and a business student. Assume we are given a list E of n employers, a list C of n computer science students, and a list B of n business students. You are also given an $n \times n \times n$ array `happy`, where `happy[i, j, k]` has the value True if employer e_i, computer science student c_j, and business student b_k are all willing to matched together and False otherwise. You want to determine whether it's possible to find n teams consisting of an employer, computer science student, and business student where everyone is matched with a team they're willing to work with, and everyone appears in exactly one team. That is, you want to find n teams $\{e_i, c_j, b_k\}$ such that each employer, computer science student, and business student appears in exactly one team and `happy[i, j, k]` is True for every team. We call this the *CO-OP3 problem.*

1. **[3 marks]** Consider the following reduction from CO-OP3 to CO-OP2:

```
Generate one CO-OP2 instance with employers E and computer science students C.
For each employer e_i and computer science student c_j, we say that e_i and c_j are
both willing to work with each other if happy[i, j, k] is True for some k.

Generate a second CO-OP2 instance with employers E and business students B.
For each employer e_i and business student b_k, we say that e_i and b_k are
both willing to work with each other if happy[i, j, k] is True for some j.

Return YES to CO-OP3 if and only if the answer to both CO-OP2 instances is YES.
```

   Give and explain a counterexample with $n = 2$ to show that this reduction is not correct.

   Suppose in a CO-OP3 instance where $n = 2$ we have [1, 2, 1], [1, 1, 2], [2, 2, 2] of happy set to true and all other entries of happy set to false. We can check that the answer to this CO-OP3 instance is NO since [2, 1, 2], [2, 2, 1] and [1, 1, 1] of happy are are false.
   However, both CO-OP2 instances will return YES. For the first CO-OP2 instance, $\{e_1, c_1\}, \{e_2, c_2\}$ satisfy the required condition as happy[1, 1, 2] and happy[2, 2, 2] are both true. For the second CO-OP2 instance, $\{e_1, b_1\}, \{e_2, b_2\}$ satisfy the condition as happy[1, 2, 1] and happy[2, 2, 2] are both true.
   Therefore the reduction is incorrect.

2. **[2 marks]** In the next few questions, you will prove that CO-OP3 is NP-complete. Begin by proving that CO-OP3 is in NP.

   Proof:

   We need to prove that a solution for the CO-OP3 question can be guessed and verified in polynomial time. We can guess a solution for a CO-OP3 instance by generating a partition of set $E \cap C \cap B$, such that there are n sets in this partition and each set contains exactly one element from E, one element from C and one element from B. It is clear that such a guess can be generated in polynomial time.
   To check if the answer is YES, we need to check if size of solution is n and for each set in our partition $\{e_i, c_j, b_k\}$, if happy[i, j, k] = TRUE. This runs in $O(n)$. Therefore the CO-OP3 problem is in NP. ■

3. **[3 marks]** For the next part of the proof that CO-OP3 is NP-complete, we need to show that CO-OP3 is NP-hard. To show this, you **must** reduce from an NP-complete problem X described in section 8.10 of the textbook. You do not need to prove correctness of your reduction (yet), but you should explain the key elements of your reduction and why your reduction runs in polynomial time. **Hint:** for one of the problems in section 8.10, this reduction is fairly straightforward. If you're doing anything extremely complicated in your reduction, it's possible that you've selected the wrong problem.

   Let X be the 3-Dimensional Matching problem as stated in chapter 8.10 of the textbook. Given an instance of X where all three sets have cardinality n, we build an instance of the CO-OP3 problem by 1) Define 3 bijective functions, $f : X \to E$; $g : Y \to C$; $h : Z \to B$ where $E = \{e_1, e_2, \cdots, e_n\}$, $C = \{c_1, c_2, \cdots, c_n\}$, $B = \{b_1, b_2, \cdots, b_n\}$. 2) Construct the $n \times n \times n$ matrix happy such that happy[i, j, k] = True if and only if $(f^{-1}(e_i), g^{-1}(c_j), h^{-1}(b_k)) \in T$.

   For the running time of the reduction we have 1) constructing 3 bijections between two sets of cardinality n runs in $O(n)$ and 2) each entry in the matrix can be assigned a truth value by searching up if the corresponding triple is in set T, and there are n^3 entries in total. Hence the reduction runs in polynomial time.

4. **[3 marks]** As the first part of proving your reduction is correct, prove that if the answer to X is YES, then the answer to the reduced CO-OP3 instance is YES.

   By the reduction given in part 3, if the answer to X is YES, then there exists n triples in T such that all elements in $X \cup Y \cup Z$ are contained in exactly one of the n triples. Let's call the set of n such triples set S.
   Then consider $S' = \{(e_i, c_j, b_k) | (f^{-1}(x), g^{-1}(y), h^{-1}(z)) \in S\}$. Since f, g, h are bijections, each employer, compsci student and business student will be in exactly one of the n triples in S'. Each triple in S', (e_i, c_j, b_k) must satisfy happy[i, j, k] = True. This means that there exists n teams satisfying the CO-OP3 instance so that the return answer is YES.

5. **[3 marks]** Prove that if the answer to the reduced CO-OP3 instance is YES, then the answer to the original X instance is YES.

   Suppose that the answer to an reduced CO-OP3 instance is YES, then there must exist a set of n teams such that for each team (e_i, c_j, b_k), we have happy[i, j, k] = True, denote this set S'.
   We can construct $S = \{(f^{-1}(e_i), g^{-1}(c_j), h^{-1}(b_k)) | (e_i, c_j, b_k) \in S'\}$. Since f, g, h are bijections, every element in $X \cup Y \cup Z$ must be in exactly of of the n triples in S. By construction of the reduction, each triple in S must be in T. Therefore, the answer to the original X instance is YES.

6. **[3 marks]** Assume that $P \neq NP$. Is it possible to generate a correct, polynomial-time reduction from CO-OP3 to CO-OP2? If yes, provide the reduction (you do not need to prove the correctness of the reduction, but you should clearly explain the key elements of the reduction and why they are there). If no, explain why no such reduction is possible.

   The answer is no. First of all, it is clear that CO-OP2 and the graph bipartite problem both have polynomial time verifiers, i.e. they are both in NP. Suppose for the sake of contradiction that there exists a correct, polynomial time reduction from CO-OP3 to CO-OP2. Then CO-OP2 must be at least as hard as CO-OP3, so it is a NP-complete problem. From quiz 5, we know that CO-OP2 can be reduced to the graph bipartite problem in polynomial time, therefore the graph bipartite problem is NP-complete. However, there are many polynomial time solution to the graph bipartite problem. Since we are assuming that $P \neq NP$, we've reached a contradiction.