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.

   At the first duel, either the living soldier wins or the zombie soldier wins. If the living soldier wins, the size of the zombie army decreases by 1 and the size of the living army stays the same; so, the probability of the living soldier winning the duel and then the living army winning the battle is equal to the probability of the living soldier winning the duel times the probability of the living army subsequently winning the battle: that is, $\alpha P(m, n-1)$. Similarly, if the zombie wins, the size of the zombie army increases by 1 and the living army decreases by 1. Because the probability of the zombie winning is $(1-\alpha)$, the probability of the zombie winning the duel and then the

living army winning the battle is equal to $(1-\alpha)P(m-1,n+1)$. Thus, the probability of m living soldiers winning against n zombies is given by the sum of these values: since {soldier wins duel then living army wins} and {zombie wins duel then living army wins} are mutually exclusive events, the probability of either event happening is given by the sum of the two probabilities. So, $P(m,n) = \alpha P(m,n-1) + (1-\alpha)P(m-1,n+1)$.

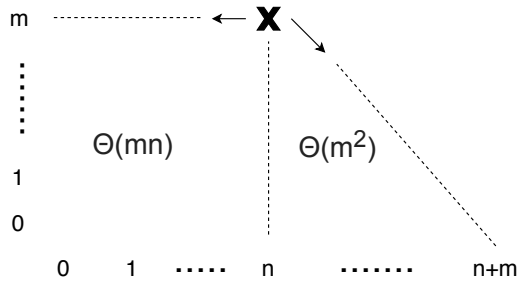
We also need to define base cases for our recurrence. If there are no living soldiers left ($m = 0$), then the probability of victory is $P(m,n) = 0$, and if there no zombies left ($n = 0$) then $P(m,n) = 1$. There's not really an obvious sensible value for $P(0,0)$ (and no other $P(i,j)$ values will depend on this value), so we choose to leave it undefined.

Putting this all together, the recurrence is:

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

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

Before we figure out an order in which to fill our table, let's figure out what table entries we need to define first. Our table is obviously going to need to have two dimensions, as our recurrence depends on two values. For simplicity, let's assume we're storing the rows and columns corresponding to $m = 0$ and $n = 0$ (it's okay if your solution doesn't do this). To compute the value at $P(m,n)$ we will obviously need all table entries $P(i,j)$ for $i \leq m$ and $j \leq n$. But because of the second recursive call we'll also need the value at $P(m-1,n+1)$; that will then require the value at $P(m-2,n+2)$, and this will continue going "down and to the right" (if we let i represent the rows of the table and j the columns[1]) until we reach the value $P(0,n+m)$ in the bottom right corner. This means our table will be in the shape of a trapezoid, with a rectangle of $(m+1)(n+1)$ entries (or, more generally, $\Theta(mn)$, depending on whether you store the zero rows and colums) next to a right triangle of $\sum_{i=1}^{m} i = \frac{m(m+1)}{2}$ entries (or, more generally, $\Theta(m^2)$, again depending on whether you've stored the zero rows and columns) – see figure below for an illustration.



When we compute the entry at $P(i,j)$, we'll need the entry to the left (at $P(i,j-1)$) and the entry below and to the right (at $P(i-1,j+1)$). This means that an acceptable ordering for the DP algorithm will be to go from the bottom row up, filling each row from left to right. Below is a DP algorithm, which initializes a 2D array of $m+1$ rows and $n+m+1$ columns, but which only fills in the entries actually used in the recurrence (i.e., for all rows except $i = 0$, we do not fill in every column, because this would require us to access table elements that are further to the right of what we actually defined in the table).

[1] This is the convention for matrices, but not for coordinates in the x-y plane. If you chose a different representation, the illustration of your table might look different, but the algorithm itself and overall space complexity should be pretty much the same.

```
Algorithm ProbVictory(m, n):
    Initialize an (m+1)x(n+m+1) 2D array P (with zero-based indexing)

    % fill in base cases
    for i=1 to m:
        P[i, 0] ← 1
    for j=1 to n+m:
        P[0, j] ← 0

    % fill in the rest of the table
    for i=1 to m:
        for j=1 to (n+m-i):
            P[i, j] ← α*P[i, j-1] + (1-α)*P[i-1, j+1]

     return P[m, n]
```

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.

   The memory use is given by the size of the table `P` which, as mentioned in the solution to the previous question, is $\Theta(mn) + \Theta(m^2)$. (Note that we cannot simplify this in general, as we don't know which term of m or n is larger.) In terms of runtime, the dynamic programming algorithm fill in $\Theta(mn) + \Theta(m^2)$ table entries (the ones illustrated in the trapezoid in the figure shown in the solution to 2.1), and computing the value in each table entry takes constant time. Therefore, the runtime is also $\Theta(mn) + \Theta(m^2)$.

   Because we will probably never need to write anything like an "explain" function for `ProbVictory` (since, in this case, the value stored at $P(m, n)$ actually **is** the "solution" we want), we could write a space-efficient implementation of `ProbVictory` that uses only $\Theta(n + m)$ memory by only keeping the current row and previous row in memory. However, this was not required for the assignment.

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).

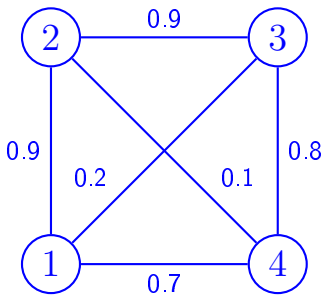
1. **[3 marks]** In class, we saw that an optimal greedy algorithm for minimizing the maximum inter-category edge weight was to initialize all photos in their own category and merge the photos adjacent to the highest-weight intercategory edge until we achieved the desired number of categories. Below

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 ≥ 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 ≤ c ≤ n
    create a list of the edges of E, in increasing order by similarity
    let C be the categorization with all photos in one category
    Num-C ← 1                ▷ Initial number of categories
    while Num-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 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 C
            Num-C ← Num-C + 1
        end if
    end while
    return 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).

Consider the following instance and let $c = 2$:



At the start of the algorithm, all four nodes are in a single category. The first edge selected is the edge between 2 and 4. At this point, the nodes 1 and 3 will be grouped with node 2 because they are both more similar to node 2 than node 4. At this point $\mathcal{C}$ has two categories, so the algorithm will terminate with the solution $\{(1,2,3),(4)\}$. The minimum weight of any intra-category edge in this solution is 0.2, between nodes 1 and 3. A **better** solution is the categorization $\{(1,2),(3,4)\}$, which has a minimum intra-category edge weight of 0.8 (between node 3 and node 4).

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

A decision version of MMCP is: given a set of n photos, a set of weighted edges E defining the similarity between each pair of photos, a desired number of categories c, **and a bound** r, does there exist a partitioning of the n photos into c categories such that no two photos in a category have a similarity less than r?

A **certificate** for this problem would be a set of categories of the photos. To verify the certificate, we need to check:

- Does the certificate contain c non-empty categories? (Takes $O(n)$ time.)
- Do all n photos appear in exactly one category? (A naïve implementation takes $O(n^2)$ time.)
- For all pairs of photos that appear within the same category, do they all have similarity greater than or equal to r? (Takes $O(n^2)$ time.)

Every step of the verification process takes polynomial time; therefore, MMCP is in NP. (It would also be acceptable to define your certificate as a partitioning of the photos, and skip the second step, or to define the certificate as a partitioning into c non-empty categories, and skip both the first and second steps.)

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.

   Notice that both problems we're dealing with produce a *partitioning* of a set of objects into a specific number of distinct sets: graph colouring aims to partition the vertices of a graph into k different colours, and MMCP aims to partition a set of photos into c different clusters. So this gives us a strong hint that we want the vertices in graph colouring to become photos in MMCP, and for the number of colours k in graph colouring to become the number of clusters c in MMCP.

   Next, we need to figure out the edge weights and r value of the reduced MMCP instance. In graph colouring, we need to make sure that vertices that share an edge don't end up being assigned the same colour. How can we "translate" that to MMCP to make sure the corresponding photos don't end up in the same cluster? Well, we can make sure that those photos have a similarity that's lower than the bounding value r. Non-connected vertices are allowed to have the same colour, so the photos corresponding to non-connected vertices should have a similarity measure that's higher than r.

   Here's a complete reduction that will do what we want:

```
Given an instance (G, k) of graph colouring:
For each vertex v_i in graph colouring, define a photo i in MMCP.

For each distinct pair of integers i, j in 1 to n:
    Define the similarity of photo i and photo j to be 0 if v_i and
    v_j share an edge in the graph colouring instance, and 1 otherwise.
    Add this value to E.

Define r = 0.99.
```

```
Return YES to graph colouring (G,k) if and only if the answer to
MMCP(n, E, k, r) is YES.
```

As for why the reduction runs in polynomial time: defining a photo for each vertex takes $O(n)$ time. Defining the similarity for each pair of photos will be $O(n^2)$ time if we can check for the existence of an edge in constant time, and in the worst case with an extremely naïve implementation might be something like $O(n^4)$ (it's all polynomial time anyway, so we don't really care about the details). All other steps of the reduction are $O(1)$, so the reduction runs in polynomial time.

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 graph colouring is YES, this means there is a way to colour the nodes in G with at most k colours such that no adjacent vertices share a colour. The obvious way to "translate" this to a certificate for the reduced MMCP instance to assign all nodes given colour i to be part of cluster i. However, a trick here is that the k-colouring of G is allowed to use **fewer than** k colours, while our solution to the reduced MMCP instance must have precisely k non-empty clusters.

   We will deal with this by converting a colouring with fewer than k colours to a (still valid) colouring that uses precisely k colours (which we know to be possible, as we've assumed that $k \leq n$). Assume the possible colours are labelled 1 to k. While there is a colour j which has not been assigned to any vertex, we know there must be some other colour i which has been assigned to two or more vertices (again, because $k \leq n$). Take one of the vertices of colour i and switch it to colour j. Continue this until every colour is assigned to at least one vertex. This will still be a valid colouring, because our initial colouring was valid, and we're just taking some vertices and assigning them a colour that isn't assigned to any other vertex (which means the swaps can't render our colouring invalid by making some vertex share a colour with a neighbour).

   Thus, the graph colouring instance has a valid colouring that uses exactly k colours, with at least one vertex assigned to each colour. We define a categorization for the reduced MMCP instance as follows: for each vertex v_i, for i from 1 to n, if v_i has colour j then we assign photo i to be in cluster j.

   We now show that this categorization is valid. Each node appears in exactly one cluster because each vertex in graph colouring was given exactly one colour, and there are k non-empty clusters because we ensured that every colour had at least one vertex assigned to it. Because the k-colouring is valid, no two vertices v_i and v_j that share a colour are connected by an edge; by our reduction, this means that any two photos i and j that are in the same cluster have a similarity equal to 1. This means all pairs of photos in the same category have a similarity greater than or equal to $r = 0.99$, which means the answer to MMCP is YES.

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.

   If the answer to MMCP is YES, this means there is a partitioning of the photos into exactly k categories such that all pairs of photos within a category have similarity equal to 1. We convert this categorization into a k-colouring for the graph colouring instance as follows: if photo i (for i in 1 to n) is in cluster j, then we assign v_i in graph colouring to have colour j.

   Because every pair of photos i, j in MMCP that share a category have a similarity equal to 1 (because these are the only similarities greater than or equal to our chosen r value of 0.99), by our reduction this means that v_i and v_j do not share an edge in the graph colouring instance. Thus, all nodes assigned to be the same colour do not share an edge, which means the converted colouring is a valid k-colouring (which uses exactly k colours, because the MMCP solution contains k non-empty clusters). Therefore, the answer to graph colouring 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.

   Consider the instance with $n = 2$, where `happy[i, j, k]` is True for `happy[1, 1, 1]`, `happy[2, 2, 1]`, and `happy[2, 1, 2]`, and False otherwise. This is a NO instance, as there is no way to form two teams $\{e_i, c_j, b_k\}$ such that each e_i, c_j, and b_k appears in exactly one team and `happy[i, j, k]` is true for both teams. (The only permissible team containing employer e_1 is $\{e_1, c_1, b_1\}$, but we can't have that as part of our solution because `happy[2, 2, 2]` is False.)

   For the first CO-OP2 instance (between employers and computer science students), we have that e_1 is willing to work with c_1 and e_2 is willing to work with c_1 and c_2. The answer to this CO-OP2 instance is YES, as we can pair e_1 with c_1 and e_2 with c_2. For the second CO-OP2 instance (between employers and business students), we have that e_1 is willing to work with b_1 and e_2 is willing to work with b_1 or b_2. The answer to this CO-OP2 instance is YES, as we can pair e_1 with b_1 and e_2 with b_2. Thus, the reduction will incorrectly return YES for this instance, even though the answer is actually NO.

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.

   We can define a certificate for CO-OP3 as a list of teams each consisting of an employer, CS student, and business student. To verify the certificate, we need to check:

- That each employer, CS student, and business student appears in exactly one team. This will take $O(n^2)$ time, or possibly less with some clever implementation details that we don't care about because $O(n^2)$ is still polynomial time.
- That `happy[i,j,k]` is True for every team e_i, c_j, b_k. This will take $O(n)$ time, assuming constant time for array lookups.

Because we can verify the certificate in polynomial time, CO-OP3 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.

   We will reduce from 3-Dimensional Matching (3DM): Given disjoint sets X, Y, and Z, each of size n, and given a set $T \subseteq X \times Y \times Z$ of ordered triples, does there exist a set of n triples in T so that each element of $X \cup Y \cup Z$ is contained in exactly one of these triples?

   As indicated in the hint, this reduction is straightforward: the employers, computer science students and business students in CO-OP3 are similar to the sets X, Y, and Z in 3DM, and our "happy teams" are similar to the elements of T. As such, we define the reduction from 3DM to CO-OP3 as follows:

```
For each x_i in X, define an employer e_i in E.
For each y_j in Y, define a computer science student c_j in C.
For each z_k in Z, define a business student b_k in B.

Define an n*n*n array happy, with all entries set to False.
For each triple (x_i, y_j, z_k) in T, set happy[i, j, k] to True.

Return YES to 3DM if and only if the answer to CO-OP3(E, C, B, happy) is YES.
```

   Every step of this reduction is in linear (and therefore, 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.

   If the answer to 3DM is YES, this means there exists a set of n triples in T such that each element of $X \cup Y \cup Z$ appears in exactly one triple. We convert this to a certificate to CO-OP3 as follows: for each triple (x_i, y_j, z_k) in the solution to 3DM, add the team (e_i, c_j, b_k) to the CO-OP3 certificate. We know that each employer, CS student, and business student will appear in exactly one team, because each element of $X \cup Y \cup Z$ appears in exactly one triple of the 3DM solution. And we know that `happy[i,j,k]` is true of every team (e_i, c_j, b_k) in our solution, because we set `happy[i,j,k]` to be True for each triple (x_i, y_j, z_k) in T. Therefore, this is a valid certificate for CO-OP3, and the answer to CO-OP3 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.

   If the answer to CO-OP3 is YES, this means there exists a set of n teams consisting of one employer, one CS student, and one business student such that everyone appears in exactly one team, and `happy[i,j,k]` is True for every team (e_i, c_j, b_k). We convert this to a certificate for 3DM as follows: for each team (e_i, c_j, b_k) in the solution to CO-OP3, add the triple (x_i, y_j, z_k) to the solution to 3DM. We know that this solution consists of n triples that are all in T, because our CO-OP3 certificate

consists of n teams, and our reduction defines `happy[i,j,k]` to be True if and only if (x_i, y_j, z_k) is in T. And we know that each element of $X \cup Y \cup Z$ appears in exactly one triple because each employer, computer science student, and business student appear in exactly one team. Therefore, this is a valid certificate for 3DM, and the answer to 3DM 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 question mentions that the tutorial quiz gives a reduction from CO-OP2 to bipartite matching. A Google search (or section 7.5 of your textbook) will reveal that bipartite matching is solvable in polynomial time – specifically, $O(mn)$ time with the Ford-Fulkerson algorithm, where m is the number of edges and n the number of vertices in the graph. (For an unweighted graph, there are other algorithms – e.g., Hopcroft-Karp – that can do this more efficiently, but the exact time complexity doesn't really matter here because we only care that the algorithm is polynomial time.) The tutorial quiz's reduction from CO-OP2 to maximum bipartite matching takes at most $O(n^2)$ time, and solving the reduced maximum bipartite matching problem will take $O(n^3)$ time in the worst case (which will happen if the graph of the reduced bipartite matching instance is fully connected – i.e., when every employer is willing to work with every student and vice versa). Therefore, CO-OP2 is in P, because it can be solved in $O(n^3)$ time in the worst case.

   This means that, if $P \neq NP$, it is **not possible** to generate a correct, polynomial-time reduction from CO-OP3 to CO-OP2. If we could reduce CO-OP3 to CO-OP2 in polynomial time, we could then solve CO-OP2 in polynomial time and therefore obtain a polynomial-time solution to CO-OP3. Because CO-OP3 is NP-complete, this cannot possibly be a correct solution if $P \neq NP$.