COSC 320: Assignment 1

This assignment is due **Friday, January 17 at 7 PM**. Late submissions will not be accepted. Please follow these guidelines:

- Prepare your solution using LaTeX or word and submit a pdf file. 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.
- Submit the assignment via Canvas.

Before we begin, a few notes on pseudocode throughout COSC 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 COSC 320 student unfamiliar with the problem you are solving. You may **neither** include what we consider to be irrelevant coding details **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...

Group Members

Please list the full names and student IDs of all group members here (even if you are submitting by yourself). We will deduct a mark if this is incorrect or missing or is a name different than the one that appears on Canvas.

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. To shade a bubble below, replace the LaTeX command `\fillinMCmath` by `\fillinMCmathsoln`. This question is worth 2 marks.

1. All group members have read and followed the guidelines for groupwork on assignments given on the syllabus (see `https://canvas.ubc.ca/courses/154976/assignments/syllabus`, under Academic Conduct -> Assignments).

   ◯ 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. One or more of us collaborated with other COSC 320 students; none of us took written notes during our consultations and we took at least a half-hour break afterwards.

   ◯ Yes ◯ No

   If yes, please list their name(s) here:

5. One or more of us collaborated with or consulted others outside of COSC 320; none of us took written notes during our consultations and we took at least a half-hour break afterwards.

   ◯ Yes ◯ No

   If yes, please list their name(s) here:

2 SMP Extreme True or False

Each of the following problems concerns a SMP scenario, and a statement about that scenario. Recall that an instance of size n of the Stable Matching Problem (SMP) has n employers and n applicants, and can be specified using $2n$ preference (or ranking) lists — one for each employer and one for each applicant.

Each statement may be **always** true, **sometimes** true, or **never** true. Select the best of these three choices and then:

- If the statement is **always** true, (a) give, and very briefly explain, an example instance in which it is true and (b) prove that it is always true.
- If the statement is **never** true, (a) give, and very briefly explain, an example instance in which it is false and (b) prove that it is always false.
- If the statement is **sometimes** true, (a) give, and very briefly explain, an example in which it is true and (b) give and very briefly explain an example instance in which it is false.

Here are the problems:

1. [2 points] Let I be any instance of SMP in which a_j is the lowest ranked applicant of employer e_i.

   Statement: If e_i and a_j are paired in *some* stable matching for I, then all stable matchings for I must have e_i and a_j paired. [Note: This statement is about *all* stable matchings, not just those produced by any specific algorithm such as the Gale-Shapley algorithm.]

2. [2 points] Let I be an instance of SMP, where the Gale-Shapley algorithm produces a stable matching for I in which all employers AND all applicants get their top-ranked choice.

   Statement: If applicant a_j is the top choice of employer e_i in instance I, employer e_i is also the top choice of applicant a_j in instance I.

3 Counting Matchings

1. [2 points] How many different SMP instances of size n are there in total? Check one, and provide a short justification of your answer.

   ◯ $n \times n!$ ◯ $(n!)^n$
   ◯ $2n \times n!$ ◯ $(n!)^{2n}$

2. [2 points] Of the total number of different SMP instances of size n, how many are such that e_i is the top choice of a_i, and a_i is the top choice of e_i, for all i from 1 to n? Check one, and provide a short justification of your answer.

   ◯ $(n-1)!$ ◯ $((n-1)!)^n$
   ◯ $2n \times (n-1)!$ ◯ $((n-1)!)^{2n}$

4 Tour Planning

You're organizing a back-to-school social tour for CS students. Because the major is extremely popular, the students will be partitioned into three groups. You want to determine whether there's a *good solution*, where no student knows anyone else in their group (other than themselves), so as to maximize the opportunity for people to meet new people. Moreover, each of the three groups should be non-empty, although groups need not be of the same size. We'll call this the TG (Tour Grouping) problem.

You have at your disposal a handy matrix $K[1..n][1..n]$, where $n \geq 3$ is the number of students. Entry $K[i, j]$ is 1 if student i knows student j, and is 0 otherwise. The matrix is symmetric, i.e., $K[i, j] = K[j, i]$. (Entry $K[i, i]$ is always 1, for $1 \leq i \leq n$.)

For example, if the number n of students is eight, and the matrix K is as on the left below, then you can put students 1, 2, and 6 in one group, students 3 and 4 in another group, and 5, 7, and 8 in the third. So for this instance of the problem, the answer is Yes (there is a good solution). However, if all entries in the matrix K are equal to 1, then there clearly is no good solution so the answer is No.

	1	2	3	4	5	6	7	8
1	1	0	0	1	1	0	0	0
2	0	1	1	1	0	0	1	0
3	0	1	1	0	1	1	1	1
4	1	1	0	1	0	0	1	1
5	1	0	1	0	1	0	0	0
6	0	0	1	0	0	1	1	0
7	0	1	1	1	0	1	1	0
8	0	0	1	1	0	0	0	1

	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1
2	1	1	0	0	0	0	0	0
3	1	0	1	0	0	0	0	0
4	1	0	0	1	0	0	0	0
5	1	0	0	0	1	0	0	0
6	1	0	0	0	0	1	0	0
7	1	0	0	0	0	0	1	0
8	1	0	0	0	0	0	0	1

1. [2 points] For the example matrix K given above on the right, where again there are eight students, give one good solution to the TG problem.

5 A Brute Force Approach to Tour Planning

Here is pseudocode for a brute force algorithm that either outputs a good solution to the TG problem described above, if one exists, or else outputs "no good solution". The algorithm generates all possible ways to group students into three groups, and checks whether any such grouping is a good solution. This brute force algorithm could use or adapt the subroutine called GENERATE-GROUPINGS to enumerate all possible ways to put the students into three groups, $G = (G_1, G_2, G_3)$.

```
1: function TOUR-PLANNING-BRUTE-FORCE(n, K[1..n][1..n])
2:     ▷ n ≥ 3 is the number of students
3:     ▷ entries of K are either 0 or 1, and K is symmetric

4:     for each possible grouping G = (G1, G2, G3) of the students into three groups do
5:         ▷ assume that the GENERATE-GROUPINGS function below is adapted to enumerate the groupings
6:         check that each of the groups is non-empty
7:         check that any two people in the same group don't know each other
8:         if both of these checks pass then
9:             return the grouping G (and halt)                    ▷ this is a good solution
10:        end if
11:    end for
12:    return "no good solution'
13: end function
14:
15: function GENERATE-GROUPINGS(n,G[1..n],i)
16:    if i = n + 1 then return G[1..n]
17:    else
18:        G[i] ← 1; GENERATE-GROUPINGS(n,G[1..n],i + 1)
19:        G[i] ← 2; GENERATE-GROUPINGS(n,G[1..n],i + 1)
20:        G[i] ← 3; GENERATE-GROUPINGS(n,G[1..n],i + 1)
21:    end if
22: end function
```

Here, a grouping of the students is represented as an array $G[1..n]$, where each entry $G[i]$ is either 1, 2, or 3, indicating which group student i is in. For example, if $n = 8$ and $G[1..n]$ is [1,1,2,2,3,1,3,3], this represents the grouping discussed in our example above, with students 1, 2, and 6 in the first group, students 3 and 4 in the second, and 5, 7, and 8 in the third. When $n = 3$, the function call GENERATE-GROUPINGS(3, $G[1..3], 1$) outputs the arrays (representing groupings) in the order

[1,1,1], [1,1,2], [1,1,3], [1,2,1], [1,2,2], [1,2,3], [1,3,1], [1,3,2], [1,3,3], [2,1,1], ...

and so on. (The array G need not be initialized for this function call.) Note that while the groupings output by GENERATE-GROUPINGS ensure that each student is in exactly one group, some groups may be empty. That is, the groupings might not be a *partition* of the students into non-empty groups. So the brute force algorithm includes a check to ensure that if a good solution is output, all groups are indeed non-empty.

1. [2 points] As a function of n, how many groupings are generated by the GENERATE-GROUPINGS algorithm? Briefly explain your answer (one short sentence).

2. [3 points] Now suppose that we change the ordering of the lines to get an alternative algorithm (which also generates all possible groupings):

```
function GENERATE-GROUPINGSS-MODIFIED(n, G[1..n],i)
    if i = n + 1 then return G[1..n]
```

else
$G[i] \leftarrow 1$; GENERATE-GROUPINGS-MODIFIED(n,$G[1..n]$,$i+1$)
$G[i] \leftarrow 3$; GENERATE-GROUPINGS-MODIFIED(n,$G[1..n]$,$i+1$)
$G[i] \leftarrow 2$; GENERATE-GROUPINGS-MODIFIED(n,$G[1..n]$,$i+1$)
end if
end function

On the call GENERATE-GROUPINGS-MODIFIED(3,$G[1..3], 1$), what is the ordering in which arrays are output? Give the first **seven** arrays in the ordering. No justification needed.

3. [4 points] Write pseudocode to check whether a particular grouping satisfies the second check of the TOUR-PLANNING-BRUTE-FORCE function, i.e., that for each group, check that no-one in the group knows anyone else in the group. The algorithm should return "Check fails" if in some group two students know each other, and otherwise should return "Check passes".

function CHECK-WHO-KNOWS-WHO(n, $G[1..n]$, $K[1..n][1..n]$)

end function

4. [2 points] Give a Θ bound on the worst case runtime of your algorithm from part 3, and provide a brief justification.