CPSC 320 2025S: Assignment 1

This assignment is due **Friday, July 11 at 7 PM**. Late submissions will not be accepted. Please follow these guidelines:

- Prepare your solution using LaTeX and submit a pdf file. Easiest will be to submit using 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.
- Submit the assignment via GradeScope. You will be get access to Gradescope on the 9th of July via an invite link. 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.

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. 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 CWLs of all group members here (even if you are submitting by yourself). We will deduct a mark if this is incorrect or missing.

- Jason Mac

1 SMP Extreme True And/Or False

Each of the following problems represents a SMP scenario (with n employers and n applicants) and a statement about that scenario. 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. [3 points] **Scenario:** an SMP instance with $n \geq 2$. **Statement:** there is exactly one stable matching.

   $\exists (e, a) \notin M$ such that e prefers a over their current and a prefers e over their current, unstable.

   This is sometimes true and sometimes false. For example consider the following instance:

   - $e_1 : a_1, a_2$
   - $e_2 : a_1, a_2$
   - $a_1 : e_1, e_2$
   - $a_2 : e_1, e_2$

   The only possible solutions would be $M_1 = \{(e_1, a_1), (e_2, a_2)\}$ and $M_2 = \{(e_1, a_2), (e_2, a_1)\}$. But M_2 is not stable since $(e_1, a_1) \notin M_2$ but e_1 prefers a_1 over a_2 and a_1 prefers e_1 over e_2. Since $G-S$ guarantees a solutino then M_1 is a solution.

   Then consider the next example consider the following instance:

   - $e_1 : a_1, a_2$
   - $e_2 : a_2, a_1$
   - $a_1 : e_2, e_1$
   - $a_2 : e_1, e_2$

   The only possible solutions for this one is $M_1 = \{(e_1, a_1), (e_2, a_2)\}$ and $M_2 = \{(e_1, a_2), (e_2, a_1)\}$.

   But in this case, both M_1, M_2 are stable.

2. [4 points] **Scenario:** an SMP instance with $n \geq 2$. **Statement:** there exists a stable matching in which nobody gets their first choice.

   This is sometimes true and sometimes false.

   Consider the following instance

   - item

3. [4 points] **Scenario:** an SMP instance with $n \geq 2$ where all employers have identical preference lists. **Statement:** there is exactly one stable matching.

   - $e_1 : a_1, a_2$
   - $e_2 : a_1, a_2$
   - $a_1 : a_1, a_2$
   - $a_2 : a_1, a_2$

   $\exists (e, a) \notin M$ such that e prefers a over their current and a prefers e over their current, unstable.

   By $G - S$ algorithm we are guaranteed one solution. Then suppose that there exists another solution M' such that $M' \neq M$.

   We know that each employer gets the best possible applicant they can get and also that the applicants will get the worse possible employer they can get.

   Then we have that for some $(e, a) \in M$ that $(e, a) \notin M'$.

   Then we get that $(e, a'), (e', a) \in M'$ for some e', a'.

   We know e and e' share the same applicant list and thus either a' ranks higher than a or the other way.

The statement is false. Take $n = 2$ and we construct the following SMP instance of 2 employees and applicants denoted by $E = \{e_1, e_2\}$ and $A = \{a_1, a_2\}$ respectively. We define this instance through the following preference lists:

- $a_1 : e_1, e_2$
- $a_2 : e_2, e_1$
- $e_1 : a_2, a_1$
- $e_2 : a_1, a_2$

We have that a stable matching can be found through the pairings, $(e_1, a_1), (e_2, a_2)$, but also with $(e_1, a_2), (e_2, a_1)$.

Now, for each $n > 2$, we construct a new instance through the following:

- $a_1 : e_1, e_2, \ldots$
- $a_2 : e_2, e_1, \ldots$
- $a_i : e_i, \ldots$
- $e_1 : a_2, a_1, \ldots$
- $e_2 : a_1, a_2, \ldots$
- $e_i : a_i, \ldots$

for $i \in \{3, 4, \ldots, n\}$, and where the permutation of the preference list does not matter except for the specified entry.

Then we have the two following stable matchings for such an instance:
$S_1 = \{(e_i, a_i) : i = 3, 4, \ldots, n\} \cup \{(e_1, a_1), (e_2, a_2)\}$, and
$S_2 = \{(e_i, a_i) : i = 3, 4, \ldots, n\} \cup \{(e_1, a_2), (e_2, a_1)\}$.

We never have an instability since by construction, for $3 \leq i \leq n$, e_i always prefers a_i over anyone else and vice versa.

In other words, not every instance for $n \geq 2$ will have exactly one stable matching, but there always exists at least one by $G - S$ algorithm.

2 Medieval Matching Problems

In the 14th century, the Bubonic plague killed between one-third and one-half of Europe's population. With much of the peasantry wiped out by the plague, there was a severe labour shortage as there were no longer enough serfs, tenant farmers, and labourers to work the existing farmland. As a positive result of all this, labourers had more say over whom they would work for, which led to increased wages, better working conditions, and, ultimately, the end of European feudalism.

Here we consider an SMP variant where we have k lords (employers), where employer i has s_i "slots" for peasant farmers (applicants), but the total number of slots $\sum_{i=1}^{k} s_i$ (which we'll denote by n) exceeds the number of workers (which we'll call m). Each employer has a complete preference list of applicants and vice versa. Moreover, every employer would rather have a slot filled by any applicant than leave a slot unfilled, since leaving a slot unfilled means land will go unmaintained and crops will rot in the fields. We call this the **Labour Shortage Matching Problem** (LSMP). A solution to LSMP requires that every applicant be matched to an employer, but not that every employer's slot be filled (since that would be impossible).

1. [1 point] We will need to expand our definition of "instability" for this problem. The definition we saw in class for RHP still applies to LSMP: namely, when e_i is matched with one or more applicants and a_j is matched with an employer, but a_j prefers e_i to their current employer and e_i prefers a_j over any one of its matched applicants.

   Describe a new type of instability that can occur between an applicant and an employer with one or more unfilled slots.

2. [1 point] Give and briefly explain a small example in which the only possible stable matching has an employer with no matches.

3. [2 points] Describe a (natural) modification of the Gale-Shapley Algorithm which could be used to solve the Labour Shortage Matching Problem (LSMP). You may use words to explain your algorithm but should also provide pseudocode. No proof of correctness is being requested. Below is the Gale-Shapley algorithm, included mainly so you'll have the LaTeX source to work with:

```
1: procedure GALE-SHAPLEY(E, A)
2:     Initialize all employers and applicants as unmatched
3:     while there is an unmatched employer with at least one applicant on its preference list do
4:         choose such an employer e ∈ E
5:         make offer to next applicant a ∈ A on e's preference list
6:         if a is unmatched then
7:             Match e with a                                  ▷ a accepts e's offer
8:         else if a prefers e to their current employer e′ then
9:             Unmatch a and e′                                ▷ a rejects e′
10:            Match e with a                                  ▷ a accepts e's offer
11:        end if
12:        cross a off e's preference list
13:    end while
14:    report the set of matched pairs as the final matching
15: end procedure
```

4. [3 points] Give a reduction from LSMP to RHP (recall that RHP is defined in worksheet 2). Remember that your reduction needs to describe both how to convert an instance of LSMP to an instance of RHP and how to convert the RHP solution back to an LSMP solution.

5. [4 points] Prove the correctness of your reduction from 2.4. In other words, prove that if the matching returned by the RHP solver is correct (i.e., is a perfect matching with no instabilities and the correct number of residents at each hospital), then the matching returned by your reduction will:

   (a) Have exactly one employer matched to every applicant and no more than s_i applicants matched to employer i (we can think of this as our definition of a "valid" solution to this version of the problem); and

   (b) Contain no instabilities.

This statement is never true.

Let $n \geq 2$, suppose we are given an instance an SMP. Denote set of employees by E and applicants by A. We assume for the sake of contradiction, there exists a matching M such that nobody gets their first choice. Then in the G-S algorithm, this implies that for each $e \in E$ they are either rejected on their first offer or their applicant to which they made their first offer to is poached by someone else.

We also observe that the top choices of each employer is not unique, otherwise each employer would have never have been rejected in their first choice and our assumption is false. Then we have that there exists $e_1, e_2 \in E$ such that their top choice of applicant is some $a \in A$. Observe that the top employer for a cannot be e_1 or e_2 since this would mean that either e_1 or e_2 is never rejected by a or have a stolen from them.

Thus, the top employer choice for a is some $e \in E$.

3 Red or Blue?

Boolean satisfiability (SAT) is, as far as Computer Scientists know, a hard problem. We will define precisely what we mean by 'hard', later on in the course. This problem is defined as follows:

> The input is a collection of m *clauses* over n boolean variables X_1, X_2, $\ldots X_n$. Each clause is a disjunction of some of the variables or their complements.
>
> The problem consists in answering the question "Is there a way to assign truth values to each variable that makes **every** clause of the instance TRUE?" (For short this is called a *valid truth assignment.*)

For instance, here is an instance of SAT made of 4 clauses over 3 variables:

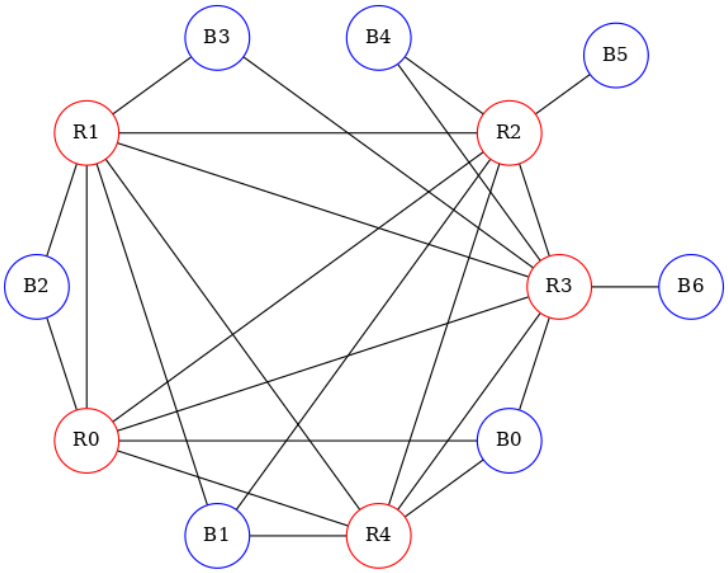
1. $X_1 \vee \overline{X}_3$
2. $\overline{X}_1 \vee \overline{X}_2 \vee X_3$
3. $\overline{X}_2 \vee \overline{X}_3$
4. X_3

Equivalently, you could think of this instance as the single boolean expression

$$(X_1 \vee \overline{X}_3) \wedge (\overline{X}_1 \vee \overline{X}_2 \vee X_3) \wedge (\overline{X}_2 \vee \overline{X}_3) \wedge X_3$$

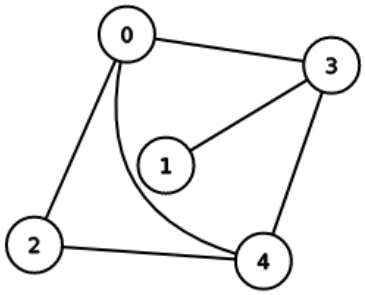
There are many other problems whose level of difficulty is similar to that of Boolean Satisfiability, and this problem has attracted a lot of research. So while the problem is difficult to solve in general, a number of tools to solve instances of the problem have been developed (see the Wikipedia page for details and links). We can take advantage of these tools to solve other problems using *reductions*.

In this question, we will look at the problem of determining whether or not a graph is a *split graph*. A graph $G = (V, E)$ is a *split graph* if we can partition its vertex set V into two subsets V_1, V_2 such that $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$, every two vertices of V_1 are connected by an edge $(\forall v \in V_1, \forall w \in V_1, \{v, w\} \in E)$ and no two vertices of V_2 are connected by an edge $(\forall v \in V_2, \forall w \in V_2, \{v, w\} \notin E)$. For instance, the following graph is a split graph, where the vertices of V_1 are colored Red, and the vertices of V_2 are colored Blue.



1. [4 points] Show how to reduce an arbitrary instance of the split graph recognition problem into an instance of *Boolean Satisfiability* (SAT). (Remember that, per the problem specifications, your clauses must be in conjunctive normal form.) Hint: you will want to introduce a single variable for each vertex of the graph.

2. [1 point] Write the collection of clauses you would get for the following graph:



3. [3 points] Explain briefly why, if an input graph is a split graph, then there is a way to assign values to the variables in the instance of SAT that makes every clause TRUE.

4. [3 points] Explain briefly why, if there is a way to assign values to the variables in the instance of SAT that makes every clause TRUE, then the graph used to generate the instance must be a split graph. Show how you would determine which vertices of G belong to which of the two subsets of V.

4 With a Little Help from my Friends

[10 points] On the popular social network Q, users "follow" other users to see the content they release. We can model this with a directed graph with a user u_i represented by a vertex, and with a directed edge (u_i, u_j) to indicate that u_i follows u_j.

We say that two users u_i and u_j are **friends** if u_i follows u_j and u_j follows u_i. We say that a group of users W form a **friend group** if every user in W is friends with every other user in W.

The **Friend Group Problem** (FGP) asks: given a directed graph $G = (U, E)$ representing the "follows" structure of our social network, does the network contain a friend group of size k?

Show how to reduce an arbitrary instance of FGP into an instance of *Boolean Satisfiability* (SAT). To save time, we are not requiring you to prove the correctness of this reduction (though it's good practice if you want to give it a try), but you must clearly explain the purpose of all the components of your reduction. Hint: my reduction uses nk variables, where n is the number of users in U.