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

$\exists (e, a) \notin M$ 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 solution then the only other possibility, M_1, is the only 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 since for each pair, at least one of the entries prefers their partners over other possibilties.

4. [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

5. [4 points] **Scenario:** an SMP instance with $n \geq 2$ where all employers have identical preference lists. **Statement:** there is exactly one stable matching.

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

   This statement is always true.

   Proof: Suppose we are given an instance with $n \geq 2$.

   By $G - S$ algorithm we are guaranteed one solution, denote it by M_{GS}.

   For sake of contradiction, we assume there exists another stable solution S such that $S \neq M_{GS}$.

   Then we have that there exists $(e, a) \in M_{GS}$ so that $(e, a) \notin S$, this is possible since $n \geq 2$.

   Now either a is the number one choice in the applicant list for all employers or not.

   - **Case 1:** Suppose that a is at the top of the applicant list.
     Since $(e, a) \notin S$ we have $(e, a_s), (e_s, a) \in S$ for some employer and applicant resepectively.
     We consider the runnings of the $G - S$ algorithm to produce M_{GS}.
     Since a is the top choice for all employers, a will receive an offer from all employers.
     This means that a rejected all other employers and accepted the offer from e.
     Which implies the number one choice of employer for a was e.
     By assumption, e_s prefers a to a_s since all employers share the same applicant list.
     Then observe that we have established the following:
     1. $(e, a) \notin S$, and $(e_s, a), (e, a_s) \in S$,
     2. e prefers a over a_s, and
     3. a prefers e over e_s.

     But by definition, this makes S unstable, which is a contradiction of the assumption.
   - **Case 2:** Suppose a is not at the top of the applicant list for all employers.
     Then we have that for some applicant a_1, that they are at the top choice for each employer.
     And we also have that $(e_1, a_1) \in M_{GS}$ for some employer e_1.
     Again, we make the same argument that a_1 must prefer e_1 over all other employers.
     Thus, both e_1 and a_1 prefer each other over all other employers and applicants.
     Observe that if $(e_1, a_1) \notin S$ then the argument from **Case 1** would lead to S not being stable.
     So then we must have that $(e_1, a_1) \in S$.

   In other words, there always exists pair (e, a) such that they prefer each other over anyone else in S.

   We know that for $n = 2$, any instance with employers that have the same applicant list, there is a unqiue solution.

Let an integer $k \geq 2$ and suppose that for such instances with $n = 2, 3, \ldots, k$ employees, that there is a unique solution.

Then consider an instance with $k + 1$ employers and applicants with identical applicant lists.

We remove the employer and applicant e_{k+1}, a_{k+1} such that they prefer each other over anyone else, and a_{k+1} is the number one applicant.

Such pairs exists for every instance size, and we will index using them the instance size.

Then for the instance with 2 employees and applicants we know there exists a unqiue solution, where it is constructed using the previous ruling.

Denote the this solution by M_2. Notice for M_2 there exists e_2, a_2 such that they prefer each other in this instance, and as well as e_1, a_1.

We then construct $M_3 := M_2 \cup \{(e_3, a_3)\}$.

And thus, $M_k = M_{k-1} \cup \{(e_k, a_k)\}$.

We have that for any e_i will always prefer their respective indexed partners, and thus

We maintain the order of the employer lists and applicant lists in this new instance.

By assumption, there exists a unique solution for this instance, denote it M_k.

Then consider $M_{k+1} := M_k \cup \{(e_1, a_1)\}$.

Observe that for any $(e, a) \in M_{k+1}$ and for any $(e, a_1), (e_1, a) \notin M_{k+1}$ that either

Then assume $(e', a') \in S$. So again, $(e, a_s), (e_s, a) \in S$ for some employer and applicant.

Then notice that $(e, a'), (e_s, a') \notin S$ given that $(e', a') \in S$.

Then either a prefers e over e_s or the other way around.

In either case, we get instability in S since $(e, a'), (e_s, a') \notin S$ and the employers prefer a'.

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.

   If there is some employee e_i with at least one free slot, and there is some applicant, a_j, such that a_j prefers e_i over their current employer, then this is unstable.

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

   Consider the instance,

   - $e_1 : a_1$
   - $e_2 : a_1$
   - $a_1 : e_1$

   Such that both employers contain one slot. We have two possible matchings (e_1, a_1) or (e_2, a_1).

   For $\{(e_2, a_1)\}$ there is incentive for a_1 to leave for e_1 since e_1 has an empty slot, which is unstable.

   So then, the solution must be $\{(e_1, a_1)\}$ which leaves e_2 with no assigned applicants.

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 employer with open slots and at least one applicant on its preference list do
4:          choose such an employer $e \in E$
5:          while $e$ has open slots do
6:              make offer to next applicant $a \in A$ on $e$'s preference list
7:              if $a$ is unmatched then
8:                  Match $e$ with $a$                                  ▷ $a$ accepts $e$'s offer
9:              else if $a$ prefers $e$ to their current employer $e'$ then
10:                 Unmatch $a$ and $e'$                                ▷ $a$ rejects $e'$
```

```
11:             Match $e$ with $a$                         ▷ $a$ accepts $e$'s offer
12:         end if
13:         cross $a$ off $e$'s preference list
14:     end while
15: end while
16: report the set of matched pairs as the final matching
17: 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.

   Denote employees by $e_1, e_2, \dots e_k$ and applicants by $a_1, a_2, \dots a_m$, so that they form a LSMP instance.

   We construct a new instance using these employees and applicants.

   Construct $n - m$ applicants, $f_{m+1}, f_{m+2}, \dots f_n$, such that their preference lists are $f : e_1, e_2, \dots e_k$.

   So then we have $(n - m) + m = n$ applicants.

   Then for each e append their preference list such that $e_{\text{new list}} : e_{\text{old list}}, f_{m+1}, f_{m+2} \dots f_n$.

   Then we have created an instance with n slots and applicants and can thus solve this using RHP.

   Given the solution S' from RHP, we convert it to LSMP by deleting each pair (e, f_i) for $m+1 \leq i \leq n$.

   Thus, the LSMP solution is $S = S' \setminus \{(e_i, f_j) : 1 \leq i \leq k, m+1 \leq j \leq n\}$.

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.

   Since RHP will return a valid solution then each real applicant must be matched with exactly one employer, and the number of applicants in an employer's slots will not exceed its capacity. Thus, when deleting the fake applicants, we can only strictly reduce the number of slots filled for any particular employer, hence the number of applicants does not exceed its capacity.

   Proof of stability: We suppose that the solution S for LSMP created through the reduction is unstable, we aim to show that S' in RHP must be unstable.

   If S is unstable, then there are two cases.

   Case 1: Assume there is some employer e_i such that all of its slots are filled and there is some applicant a_j so that e_i prefers a_j over any of its assigned applicants and a_j prefers e_i over its assigned employer.

   Notice that each of these are real applicants, otherwise e_i would have a free slot from deleting fake applicants.

   For some applicant that e_i does not prefer a_j over denote them by a'.

   Similary denote the current employer of a_j by e'. And so $(e', a_j), (e_i, a') \in S$, but $(e_i, a_j) \notin S$.

   Since none of these applicants are fake, we must have that $(e', a_j), (e_i, a') \in S'$, similarly, $(e_i, a_j) \notin S'$.

   The applicant list of e_i remains the same in RHP except that we have appended fake applicants and it's slots are still filled. Similarly, the preference list of a_j remains unchanged, by construction.

   Hence, it still remains that e_i prefers a_j over a', and a_j prefers e_i over e'.

Then the solution S' is unstable.

Case 2: Now assume that some employer e_i with a free slot, but there is some applicant a_j such that they prefer e_i over their current employer e'.

This means that for some fake applicant(s), f_p, that (e_i, f_p) was deleted from S' to create S.

Then within RHP, we must have that e_i is full and that it is assigned to least one fake applicant f_p.

So, if $(e', a_j) \in S$ then $(e', a_j) \in S'$ since it is a real applicant.

And we have that $(e_i, a_j) \notin S'$ since $(e_i, a_j) \notin S$.

By construction, we have that e_i prefers a_j over f_p but also that a_j prefers e_i over e'.

Hence, S' is unstable.

This concludes the proof of the correctness of the reduction.

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, $\dots X_n$. 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:

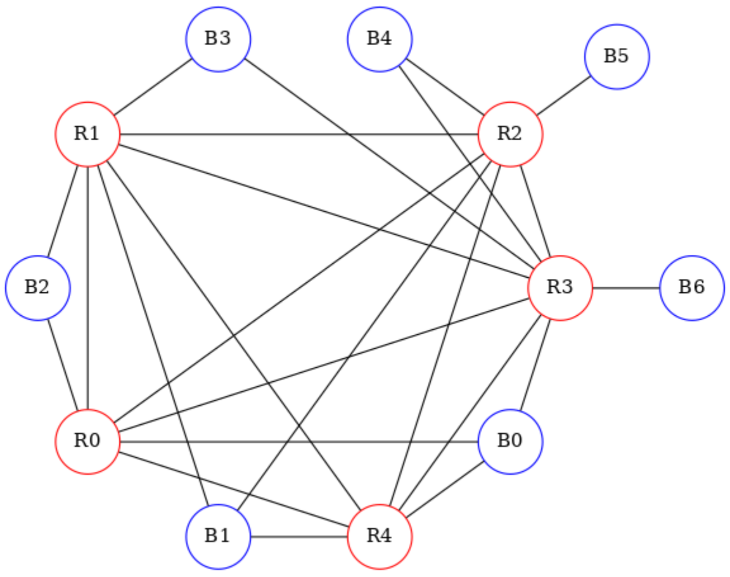
1. $X_1 \vee \overline{X}_3$
2. $\overline{X}_1 \vee \overline{X}_2 \vee X_3$
3. $\overline{X}_2 \vee \overline{X}_3$
4. X_3

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

$$\begin{array}{l}(X_1 \vee \overline{X}_3) \wedge \\ (\overline{X}_1 \vee \overline{X}_2 \vee X_3) \wedge \\ (\overline{X}_2 \vee \overline{X}_3) \wedge \\ X_3\end{array}$$

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.

   We have for any $v_i \in V$ that either $v_i \in V_1$ or $v_i \in V_2$ but not both since they form a partition over V.

   Then denote the boolean variable X_i so that

   $$X_i = \begin{cases} T, & \text{if } v_i \in V_1 \\ F, & \text{if } v_i \in V_2 \end{cases}$$

   Observe for two any vertices $\{v_i, v_j\}$ they form an edge or do not.

   We see if $v_i, v_j \in V_1$ then they share an edge, similarly, if $v_i, v_j \in V_2$ then they do not share an edge.

   If two vertices share an edge, we cannot tell if both are in V_1 but we know at least one is.

   Hence, for for ever edge pair, $\{v_i, v_j\}$ the disjunction is true, $(X_i \vee X_j)$.

   Now consider pairs $\{v_i, v_j\} \notin E$, observe if pairs do not share an edge, then both are not in V_1.

   In particular, the statement $\neg(X_i \wedge X_j)$ is true for every non-edge pair, equivalently, $(\neg X_i \vee \neg X_j)$ holds.

   Then we construct the SAT instance,

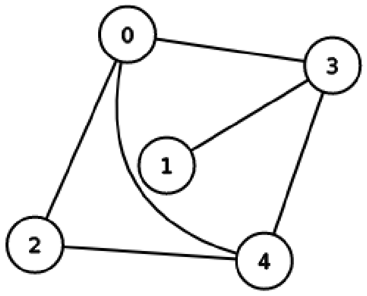
   $$\left(\bigwedge_{\{v_i, v_j\} \notin E} (\neg X_i \vee \neg X_j) \right) \wedge \left(\bigwedge_{\{v_i, v_j\} \in E} (X_i \vee X_j) \right).$$

   If such a solution to this SAT instance exists, then G is a split graph.

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

   $$(\neg X_0 \vee \neg X_1), \quad (\neg X_1 \vee \neg X_4), \quad (\neg X_1 \vee \neg X_2), \quad (\neg X_2 \vee \neg X_3),$$

   $$(X_0 \vee X_2), \quad (X_0 \vee X_3), \quad (X_0 \vee X_4), \quad (X_1 \vee X_3), \quad (X_2 \vee X_4), \quad (X_3 \vee X_4)$$



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.

   If $G = (V, E)$ is a split graph then let V_1, V_2 be the respective partition as defined.

   We assign for each $i \in V$,

$$X_i = \begin{cases} T & \text{if } i \in V_1 \\ F & \text{if } i \in V_2 \end{cases}$$

   Having that V_1, V_2 form a partition, this boolean variable is well defined.

   For each $\{i, j\}$ either they form an edge or not. If they form an edge, then both cannot be in V_2.

   Hence at least one is in V_1, and thus, the clause $X_i \vee X_j$ is true.

   If they are not an edge, then both are not in V_1. Thus, at least one is in V_2. Hence, $\neg X_i \vee \neg X_j$ is true.

   Having considered every possible clause in the reduced SAT instance, we have shown this is a satisfying truth assigment, and thus the SAT is satisfiable.

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.

   Suppose that we have a satisfying truth assingment to the SAT.

   Then for each assignment X_i, if X_i is true, we place vertex i in V_1 and if false, we place it into V_2.

   We now aim to show this makes G a split graph.

   Suppose that $i, j \in V_1$ this means that both of X_i, X_j were true.

   Hence,

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.