# Introduction

In this lab we are solving constraint satisfaction problems. There is a total of four tasks, we have to formulate the tasks as CSPs and explore the possible solutions. To create and solve the CSPs, we'll be using a software named "applet". Moreover, to the run the applet we need to setup Java Runtime Environment (JRE).

## Problem Statement 1 (Eating Out)

## Analysis:

The problem is about determining the food choices of four individuals, based on certain preferences and constraints. The problem requires finding a combination of food choices that satisfies each person's preferences and unary and binary constraints all together. We have to consider several combinations and also ensure that each person's preferences and constraints are met while formulating the CSP.

## Explanation:

First, we have to formulate the CSP before finding a solution for it. To formulate the CSP we need to define the variables, domains and constraints.

**Variables**: {Z, I, F, N}. Here, Zahid = Z, Ishrak = I, Farabi = F, Nafisa = N

**Domain**: {S, B, K, P}. Here, Special Rice, Biriyani Rice, Kashmiri Naan, and Paratha are represented as S, B, K, P respectively.

The preferences are given in the question. These preferences can be treated as constraints or the preferences can help us to figure out the constraints. Each constraint can be written as a unary or binary constraint. From the given task we can come up with these constraints:

**Constraints**:

{ (Z != [P]),

(I != F),

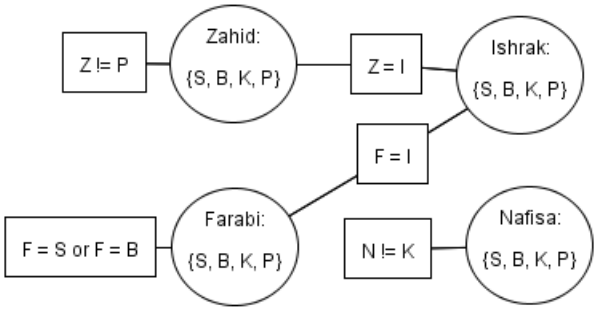
(F = [S] or F = [B]),

(Z = I), (N != [K]) }

After putting in all the unary and binary constraints into the graph, the software automatically finds a solution for the CSP. The solution that I found:

**Solution**: { (Z = [B]), (I = [B]), (F = [S]), (N = [S]) }

The Graph:




# Challenges:

Since I haven't used this software before, it took me a bit of time to work my way around the functionalities and how they work. Other than that, I found the question and the preferences quite straightforward.

# Findings:

There's also a feature named "Auto arc-consistency" which makes the arcs consistent by repeatedly removing unsupported values from the domains of the variables. This makes solving the CSP faster. We don't have to shrink the domains of the variables the software automatically finds a solution just with the help of the constraints that we mentioned.

At first, it gives us one solution, if there's more solutions than we can press the "AutoSolve" button again to find another solution. For example,
{ (Z = [B]), (I = [B]), (F = [S]), (N = [P]) } is another solution of the CSP. We can keep on producing alternative solutions as long as there are more available solutions.

# Problem Statement 2 (Finding Houses)

## Analysis:

The problem is about assigning four individuals to three different floors of a building in such a way so that it follows the specific constraints mentioned. The objective is to find a valid assignment of individuals to floors that satisfies all constraints. We have to consider that several unary and binary constraints are met while formulating the CSP.

## Explanation:

To formulate the CSP we need to define the variables, domains and the constraints.

**Variables**: {A, S, M, R}. Here, Ali = A, Sristy = S, Maliha = M, Rafid = R

**Domain**: {1, 2, 3}. Here, first floor, second floor, third floor are represented as 1, 2, 3 respectively.

The constraints are given in the question. These constraints have to be satisfied for finding a valid assignment of the variables. Each constraint can be written as a unary or binary constraint. From the given task we can come up with these constraints:

**Constraints**:

{ (A != S),

(if A = M then A = M = [2]),

(if A = M then A=[3] or M=[3]),

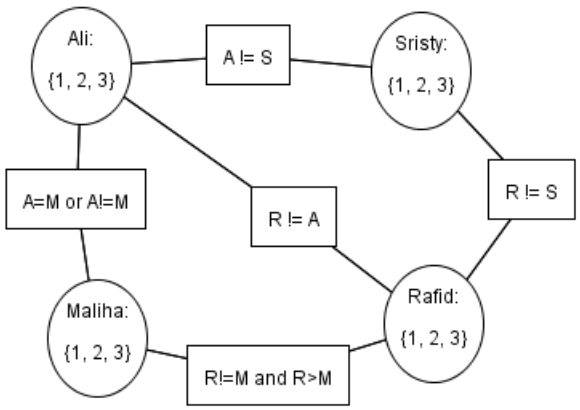
(R != A and R != S and R != M),

(R > M) }

After putting in all the unary and binary constraints into the graph, the software automatically finds a solution for the CSP. The solution that I found:

**Solution**: { (A = [2]), (M = [2]), (S = [1]), (R = [3]) }

The graph:




# Challenges:

After doing the first task, I felt a bit more comfortable using the software. As the task was pretty straightforward, I didn't face any significant challenges.

# Findings:

In this task, the domains were represented by integers rather than strings. This choice allowed us to incorporate logical constraints such as greater than and less than, which helped us in ensuring valid assignments of the variables.

Just like the previous task, “Auto Arc-Consistency” makes the arcs consistent by repeatedly removing unsupported values from the domains of the variables. This makes solving the CSP faster. Again, if there’s more solutions than we can press the “AutoSolve” button again to find another solution. For example, { (A = [3]), (M = [1]), (S = [1]), (R = [2]) } is another solution of the CSP. We can keep on producing alternative solutions as long as there are more available solutions. In this case, there are only two solutions.

# Problem Statement 3 (Spots)

## Analysis:

The problem statement at hand involves assigning each individual in a unique place in the queue. Each of the individuals will occupy a unique place inside the queue based on a set of unary and binary constraints mentioned in the question. We need to figure out the valid positions for the individuals inside the queue.

# Explanation:

To formulate the CSP we need to define the variables, domains and the constraints.

**Variables**: {R, A, F, I, T, S}. Here, Rafid = R, Atiq = A, Farhan = F, Ishmam = I, Tabassum = T, Sabrina = S

**Domain**: {1, 2, 3, 4, 5, 6}. Here, each number represents a particular position in the queue.

The constraints are given in the question. These constraints have to be satisfied for finding a valid assignment of the variables. Each constraint can be written as a unary or binary constraint. From the given task we can come up with these constraints:

**Constraints**:

{ (|F-A| = 1 and |F-I| = 1) – Since Farhan has to be between Atiq and Ishmam,

(|S-R| = 1) – Since Sabrina and Rafid are standing next to each other,

(T = 1 or T = 6) – Since Tabassum is either at the end or at the beginning,

(S = 5) – Since she has only one person behind her,

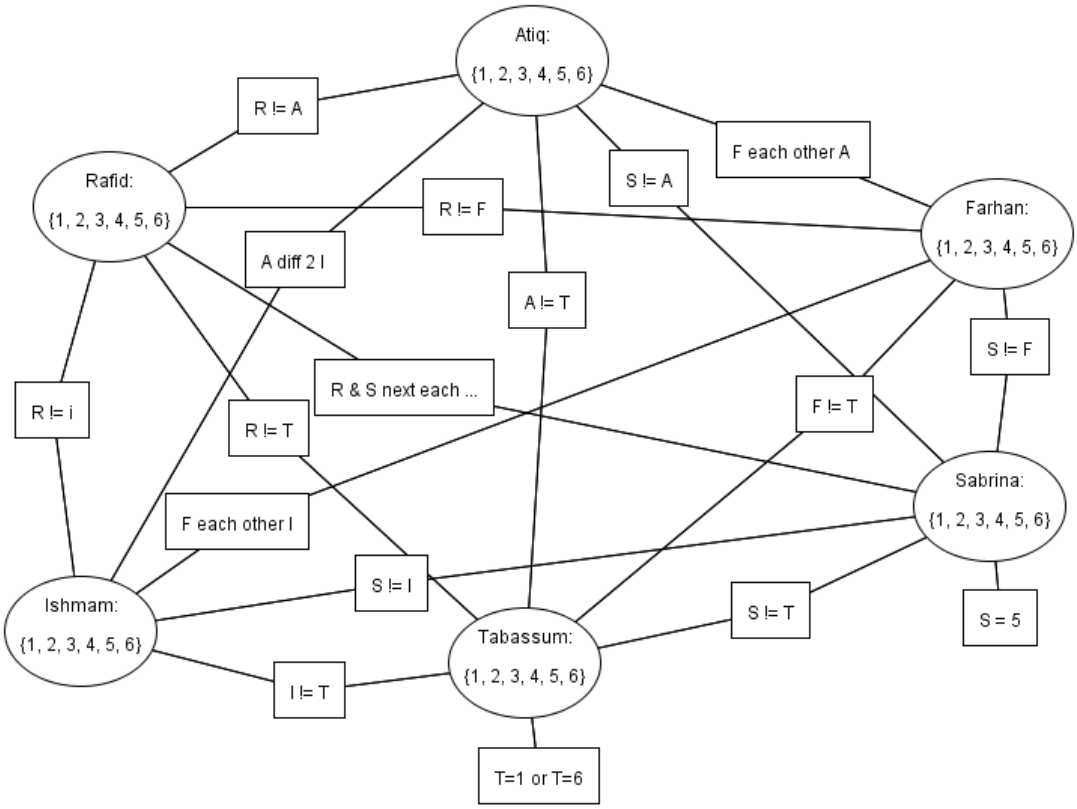
(All Diff) – Since every individual has to be positioned at a unique place. No two assignments can be the same }

The All Diff constraint could look something like this: { (R != T), (R != A), … , (F != T), (I != S), … } etc. There would be all diff constraints among the 6 variables since one's place would be unique in the queue.

After putting in all the unary and binary constraints into the graph, the software automatically finds a solution for the CSP. The solution that I found:

**Solution**: { (R = [4]), (A = [1]), (F = [2]), (I = [3]), (T = [6]), (S = [5]) }

The Graph:



# Challenges:

Took me a bit of time to figure out how to implement the constraint where Farhan had to be between Atiq and Ishmam. Therefore, the difference between Atiq and Farhan would always be two. Since there were a lot of All Diff constraints, it was easily possible for me to mistakenly omit any of the constraints. Later had to add those missing out constraints so that we can get a valid solution.

Other than that, implementing all of the constraints were time consuming and I found the whole process error prone since there was a lot of constraints to deal with.

# Findings:

Unlike the previous two tasks, this task had the All Diff constraint. This All Diff constraint ensures that all of the individuals are positioned in a unique place in the queue.

Moreover, just like the previous tasks, “Auto Arc-Consistency” and “AutoSolve” can be used if one wishes to use it. More solutions of the problem statement could be for example,
i) Rafid = 6, Atiq = 4, Farhan = 3, Ishmam = 2, Tabassum = 1, Sabrina = 5
ii) Rafid = 6, Atiq = 2, Farhan = 3, Ishmam = 4, Tabassum = 1, Sabrina = 5 and so on. We can keep on producing alternative solutions as long as there are more solutions available.

# Problem Statement 4

## Analysis:

The problem statement is about preparing a schedule for two faculty members within a fixed set of one-hour timeslots. The question provided constraints that will work as rules for task sequences, assignments tasks to faculty members, and the mutual exclusivity of lab sessions.

## Explanation:

To formulate the CSP we need to define the variables, domains and constraints.

There are in total of 4 time slots between 8am to 12pm and there two faculty members. So, we can set four variables for each faculty members – one variable for each time slot for each faculty member. This results into eight variables. Those are:

**Variables**: {X8, X9, X10, X11, Y8, Y9, Y10, Y11}. Here, X8 indicates the 8am time slot for the faculty member X. Other variables are named following the same logic.

As for the domain, the tasks can be regarded as part of it. So, the domain looks like this:

**Domain**: {G, Q, C, D, L}. Here, each letter represents a task according to the question.

The constraints are given in the question. These constraints have to be satisfied for finding a valid assignment of the variables. Each constraint can be written as a unary, binary constraint or ternary constraint. From the given task we can come up with these constraints:

**Constraints**:

{ (X11 != [Q] and X11 != [L] and Y11 != [Q] and Y11 != [L]) – Since quiz checking and AI lab takes 2 hours. So, we can't fit them in this time slot,

(X10 != [D] and X11 != [D] and Y10 != [D] and Y11 != [D]) – Since DBMS lab must finish before 10am,

(X11 != [C] and X10 != [C] and Y11 != [C] and Y11 != [C]) – Since AI class has to happen before AI lab and the last slot for AI lab is 10pm since it takes 2 hours to complete,

(Y8 != [G] and Y9 != [G] and Y10 != [G] and Y11 != [G] and X8 = [G] and X9 = [G]) – Since faculty X will gather contents not faculty Y,

(X10 != [G] and X11 != [G]) – Since X needs to gather content before the DBMS lab and the DBMS lab needs to finish before 10pm,

(X8 != [D] and X8 != [L] and Y8 != [D] and Y8 != [L]) – Since gathering content has to happen before DBMS lab and AI class needs to happen before AI lab,

(X8, X9, Y9 will have a ternary constraint, X9 and Y9 will have a binary constraint) – Since the DBMS lab has to finish by 10pm, this constraint will determine in which slot gathering contents will happen. After that we'll be determining that in which slot faculty Y has to attend the DBMS lab,

(X8, X10 and Y8, Y10 will have binary constraints) – Since the faculty that took the AI class will have to take the AI lab and the faculty who gathered content will have to take the DBMS lab, these constraints will determine in which faculty will do which task in which time slot,

(All Diff) – Since at any given time, each faculty member's allowed to do exactly one task }

The All Diff constraint could look something like this: { (X8 != X9), (Y10 != Y11), … , (X9 != Y10), (Y8 != Y11), … } etc. There would be all diff constraints among the variables since at once, each faculty member' s allowed to do exactly one task.

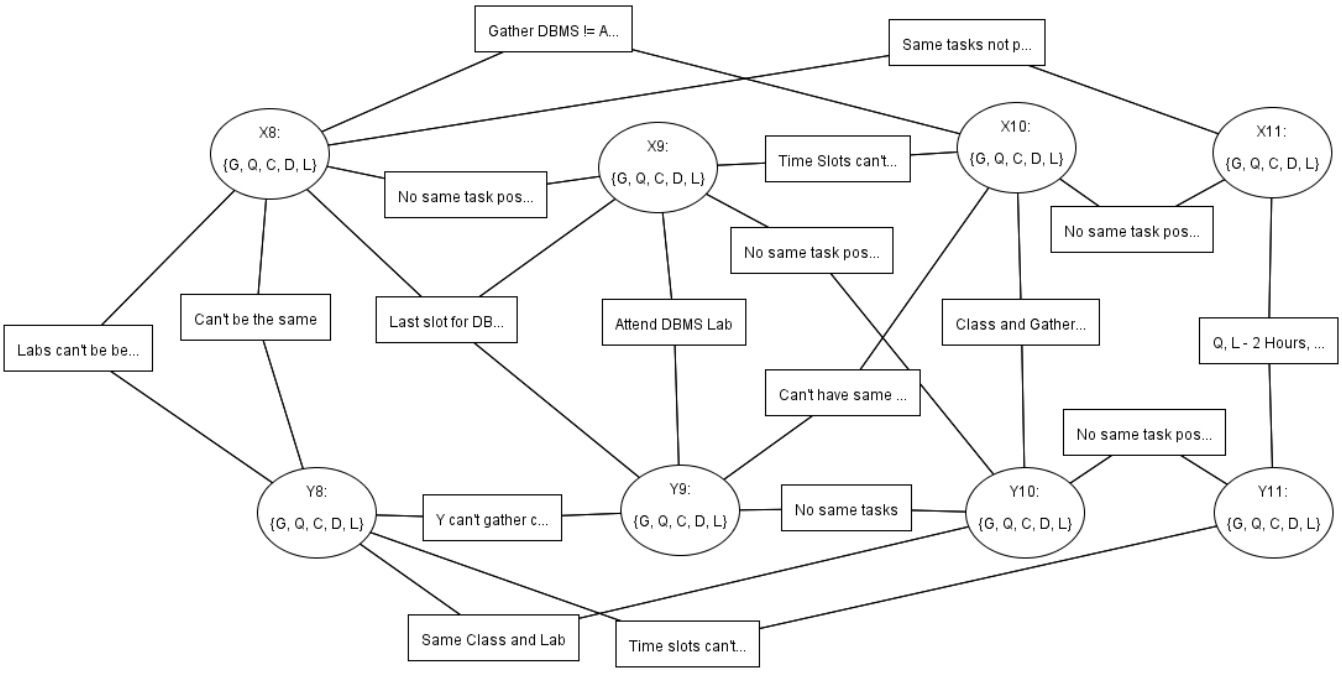
After putting in all the unary, binary, ternary constraints into the graph, the software automatically finds a solution for the CSP. The solution that I found:

**Solution**: { (X8 = [G]), (X9 = [D]), (X10 = [Q]), (X11 = []), (Y8 = [C]), (Y9 = [D]), (Y10 = [L]), (Y11 = []) }

Here, X11 and Y11 doesn't have any values. That means no task can be done in these time slots. The time slot being 11am for both faculties.

After establishing all of the unary constraints and the binary constraints other than the All Diff constraint, it shows us that the domains of the variables shrink significantly. This leads to having a lot of All Diff constraints being redundant. That is why the graph had much less All Diff constraints. This was done so that a more simplified graph can be shown for better readability.

The Graph:



# Challenges:

This is the first task where I've encountered a ternary constraint. So, I had to go through all 125 entries of that constraint to make sure each true combination is ticked and the false ones are not ticked.

Additionally, the task involved several constraints and intricate relationships between variables, making it challenging to establish suitable connections. It took good bit of time and effort to fully grasp how the variables would interact with each other based on the imposed constraints.

# Findings:

When a faculty member is conducting the DBMS lab, the other faculty must attend it and during this time this other faculty member can't do anything. To satisfy this constraint, ternary constraint was introduced among three variables.

X11 and Y11 didn't have any values assigned to them. Which meant that no faculty members can start doing any task after 11am.

Moreover, just like the previous tasks, "Auto Arc-Consistency" and "AutoSolve" can be used if one wishes to use it. In this case, there is only one solution.