CS210 : ARTIFICIAL INTELLIGENCE LAB

LAB ASSIGNMENT 7: AI & Python

Submitted By:

Name: AKSHAT SAHU

Roll No: U22CS034

Branch: CSE

Semester: 4th Sem

Division : A

Submitted To: Dr. Chandra Prakash

Department of Computer Science and Engineering

SV NATIONAL INSTITUTE OF TECHNOLOGY SURAT

2024

PART A: Exposition Problems

1. Classics Example of Logical Programming

• Movie database : provides a couple of thousands of facts about movies for you to query.
• Eliza: implements the classical shrink.
• Expert system: illustrates simple meta-interpretation of rules and asking for missing knowledge.

[Visit : https://swish.swi-prolog.org/ and the Google drive link for.pl files]

PART B : Conceptual Questions

2. Run the sample example given.

a) discuss the sample example given of Sam’s likes and dislikes in food.

- Sam likes various types of food, including Indian, Chinese, and Italian cuisines. ● Specifically, he likes mild Indian dishes such as dahl, tandoori, and kurma.
- He also enjoys Chinese dishes like chow mein, chop suey, and sweet and sour.
- Additionally, Sam likes Italian food, including pizza and spaghetti.
- Chips are another food item that Sam likes, though the type of chips is not specified.
- Sam's food preferences are expressed through a set of rules and facts in Prolog.

b) Identify the facts in the sample example. Facts:

- `indian(curry).` , `indian(dahl).` , `indian(tandoori).` , `indian(kurma).` : Facts about Indian dishes.
- `mild(dahl).` , `mild(tandoori).` , `mild(kurma).` : Facts about mild Indian dishes.
- `chinese(chow_mein).` , `chinese(chop_suey).` , `chinese(sweet_and_sour).` : Facts about Chinese dishes.
- `italian(pizza).` , `italian(spaghetti).` : Facts about Italian dishes.
- `likes(sam, chips).` : Sam likes chips.

c) Identify the rules in the taken example.

Rules:

- `likes(sam, Food) :- indian(Food), mild(Food).` : Sam likes Indian dishes that are mild.

- `likes(sam, Food) :- chinese(Food).` : Sam likes Chinese dishes.
- `likes(sam, Food) :- italian(Food).` : Sam likes Italian dishes.

d) Run what does Sam likes.

```
likes(sam, What).
```

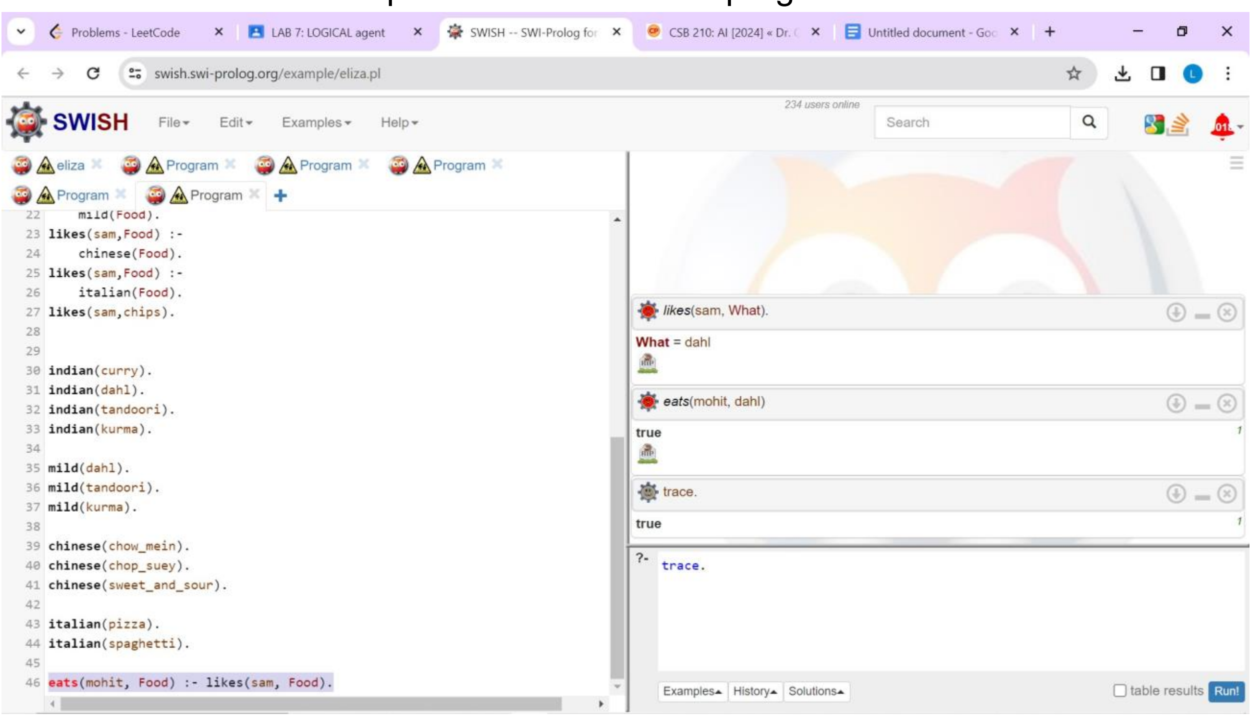
e) Sam likes curry.

```
likes(sam, curry).
```

f) Add a new rule that Mohit eat whatever Sam likes.

eats(mohit, Food) :- likes(sam, Food).

g) Tracing the execution of a Prolog query allows you to see all of the goals that are executed as part of the query, in sequence, along with whether or not they succeed. Show the steps occur in the above program. Trace.



3. Consider the following Knowledge Base:

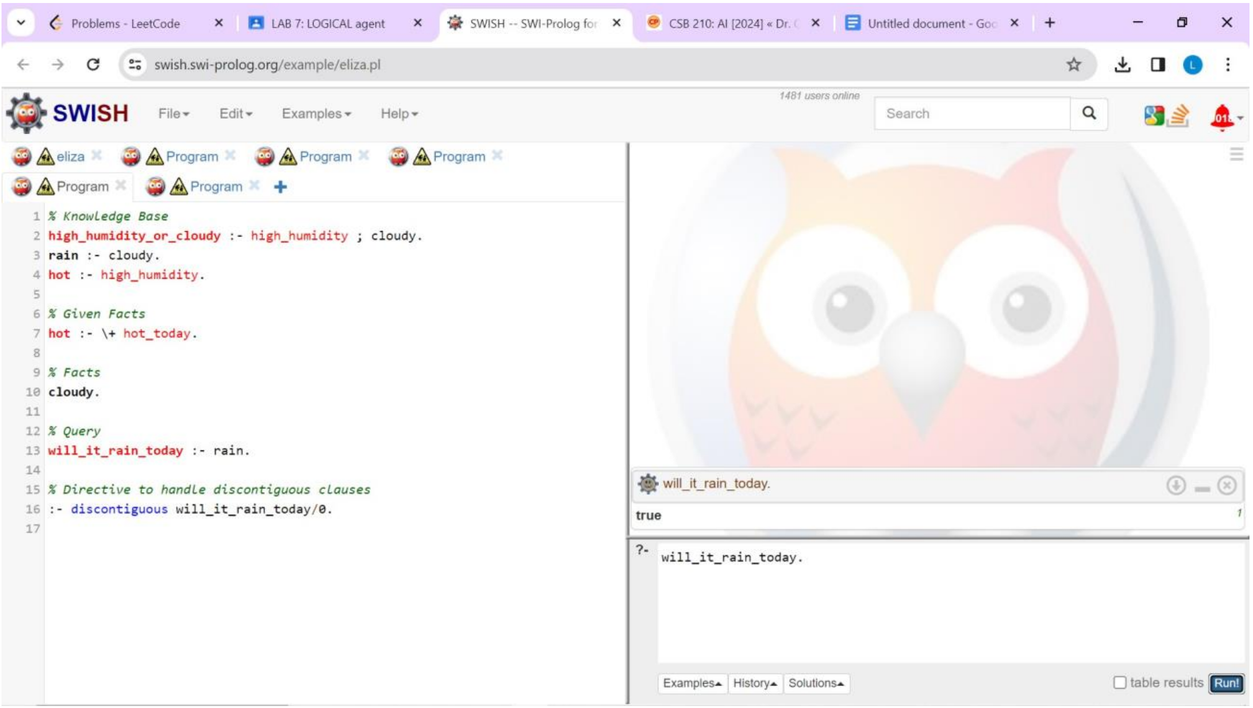
The humidity is high or the sky is cloudy.

If the sky is cloudy, then it will rain.

If the humidity is high, then it is hot.

It is not hot today.

Query : Will it rain today ?



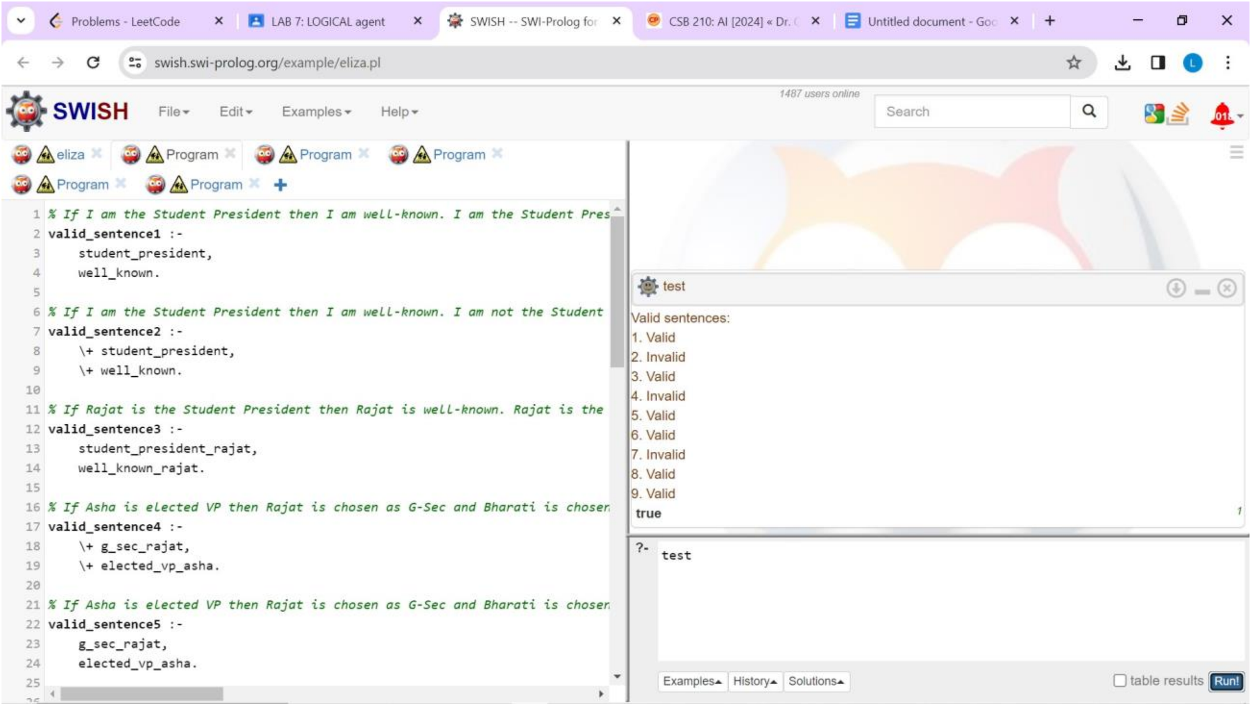
4. Write prolog program to find if the given sentences is valid or not:

- If I am the Student President then I am well-known. I am the Student President. So I am wellknown.
- If I am the Student President then I am well-known. I am not the Student President. So I am not well-known.
- If Rajat is the Student President then Rajat is well-known. Rajat is the Student President. So Rajat is well known.
- If Asha is elected VP then Rajat is chosen as G-Sec and Bharati is chosen as Treasurer. Rajat is not chosen as G-Sec. Therefore Asha is not elected VP.
- If Asha is elected VP then Rajat is chosen as G-Sec and Bharati is chosen as Treasurer. Rajat is chosen as G-Sec. Therefore Asha is elected VP.
- Wherever Mary goes, so does the Lamb. Mary goes to School. So the Lamb goes to School.
- No contractors are dependable. Some engineers are contractors. Therefore some engineers are not dependable.
- Every passenger is either in first class or second class. Each passenger is in second class if and only if the passenger is not wealthy. Some passengers are wealthy. Not all passengers are wealthy. Therefore some passengers are in second class.

- All dancers are graceful. Ayesha is a student. Ayesha is a dancer. Therefore some student is graceful.

```
% If I am the Student President then I am well-known. I am the Student President. So I am well-known.
valid_sentence1 :-
   student_president,
   well_known.

% If I am the Student President then I am well-known. I am not the Student President. So I am not  well-known.
valid_sentence2 :-
    \+ student_president,
   \+ well_known.

% If Rajat is the Student President then Rajat is well-known. Rajat is the Student President. So Rajat is well known.
valid_sentence3 :-
   student_president_rajat,
   well_known_rajat.

% If Asha is elected VP then Rajat is chosen as G-Sec and Bharati is chosen as Treasurer. Rajat is not chosen as G-Sec. Therefore Asha is not elected VP.
valid_sentence4 :-
    \+ g_sec_rajat,  \+
   elected_vp_asha.

% If Asha is elected VP then Rajat is chosen as G-Sec and Bharati is chosen as Treasurer. Rajat is chosen as G-Sec. Therefore Asha is elected VP.
valid_sentence5 :-
   g_sec_rajat,
   elected_vp_asha.

% Wherever Mary goes, so does the Lamb. Mary goes to School. So the Lamb goes to School.
valid_sentence6 :-
   goes_to_school_lamb.

% No contractors are dependable. Some engineers are contractors. Therefore some engineers are not dependable.
valid_sentence7 :-
    \+ dependable_contractors,
   \+
   engineers_not_dependable.

% Every passenger is either in first class or second class. Each passenger is in second class if and only if  the passenger is not wealthy. Some passengers are wealthy. Not all passengers are wealthy. Therefore some passengers are in second class.
valid_sentence8 :-
   passengers_second_class.
```

```
% All dancers are graceful. Ayesha is a student. Ayesha is a dancer. Therefore some student is graceful.
valid_sentence9 :-
   ayesha_dancer.

% Facts
student_president.
well_known.

student_president_rajat.
well_known_rajat.

g_sec_rajat.

elected_vp_asha.

goes_to_school_lamb.

dependable_contractors.

engineers_not_dependable.

passengers_second_class.

ayesha_dancer.

% Test predicates
test :-  write('Valid
   sentences:'), nl,
    (valid_sentence1 -> write('1. Valid') ; write('1. Invalid')), nl,
    (valid_sentence2 -> write('2. Valid') ; write('2. Invalid')), nl,
    (valid_sentence3 -> write('3. Valid') ; write('3. Invalid')), nl,
    (valid_sentence4 -> write('4. Valid') ; write('4. Invalid')), nl,
    (valid_sentence5 -> write('5. Valid') ; write('5. Invalid')), nl,
    (valid_sentence6 -> write('6. Valid') ; write('6. Invalid')), nl,
    (valid_sentence7 -> write('7. Valid') ; write('7. Invalid')), nl,
    (valid_sentence8 -> write('8. Valid') ; write('8. Invalid')), nl,
   (valid_sentence9 -> write('9. Valid') ; write('9. Invalid')), nl.
```



5. Construct your family tree diagram (start from grandparents to your siblings). and formulate defi-nitions for a human family tree using relations 'male', 'female', 'parent', 'father', 'mother', 'sibling','grandparent', 'grandmother', 'grandfather', 'cousin', 'aunt', and 'uncle'. Let 'male', 'female', 'parent' be the fundamental relations and define the others in terms of these. Write your information in facts in English.

% Fundamental relations
male(grandfather).
male(father).
male(uncle).
male(cousin1).
male(cousin2).

female(grandmother).
female(mother).
female(aunt).
female(sister).
female(cousin3).
female(me).

parent(grandfather, father).
parent(grandfather, uncle).
parent(grandmother, father).
parent(grandmother, uncle).
parent(father, me). parent(father, sister).

parent(uncle, cousin1). parent(uncle, cousin2). parent(uncle, cousin3). parent(mother, me). parent(mother, sister). parent(aunt, cousin1). parent(aunt, cousin2). parent(aunt, cousin3).

% Derived relations father(X, Y) :- male(X),

parent(X, Y). mother(X, Y) :- female(X),

parent(X, Y). sibling(X, Y) :- parent(Z, X),

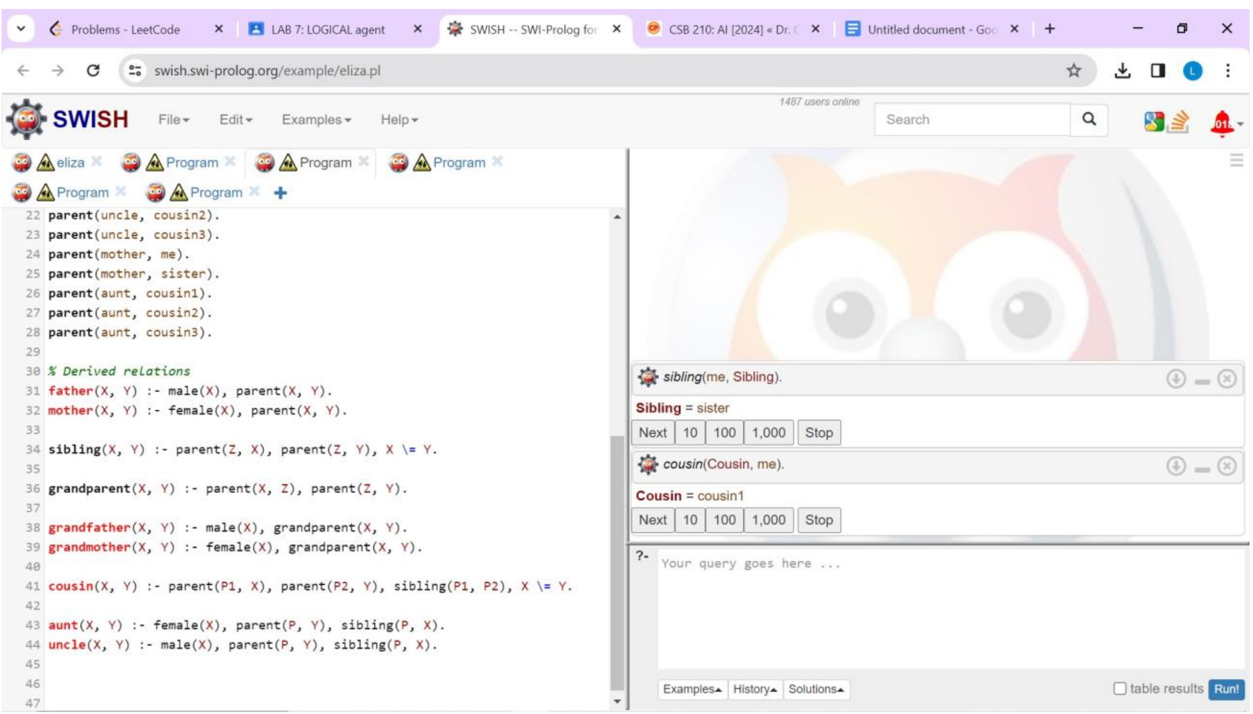
parent(Z, Y), X \= Y. grandparent(X, Y) :-

parent(X, Z), parent(Z, Y).

grandfather(X, Y) :- male(X), grandparent(X, Y).
grandmother(X, Y) :- female(X), grandparent(X, Y).

cousin(X, Y) :- parent(P1, X), parent(P2, Y), sibling(P1, P2), X \= Y.

aunt(X, Y) :- female(X), parent(P, Y), sibling(P, X).
uncle(X, Y) :- male(X), parent(P, Y), sibling(P, X).



6. Consider the following facts/statements. The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

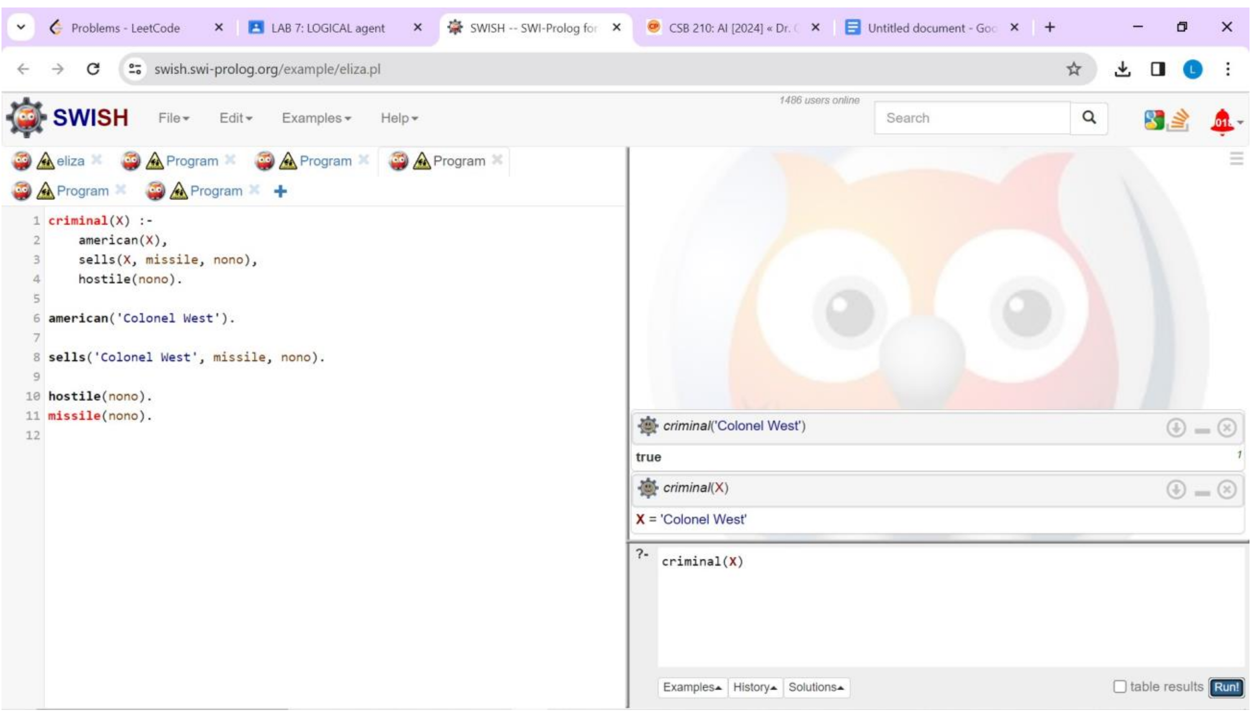
Formulate this knowledge in First Order Logic. And use prolog program to execute following queries: a)Query : criminal(west)?
b)Query: criminial(X)?
Draw a resolution tree to find the answer of par (a)

criminal(X) :-
  american(X), sells(X,
  missile, nono),
  hostile(nono).

american('Colonel West').

sells('Colonel West', missile, nono).

hostile(nono).
missile(nono).



PART C : Exploratory Problem [10 MARKS]

7. There is a famous problem in mathematics for coloring adjacent planar regions. Like cartographic maps, it is required that, whatever colors are used, no two adjacent regions may not have the same color. Two regions are considered adjacent provided they share some boundary line segment. Consider the following map.

Develop a Prolog program that can compute all possible colorings (Given colors to color with) are Red

,Blue, Green and Yellow. [Hint : Covert it to graph first]

% Define the map as a graph where each node represents a region
% and each edge represents adjacency between regions.
adjacent(1, 2). % Example adjacency edges, you need to define all adjacencies for your map adjacent(1, 3). adjacent(1, 4). adjacent(1, 6). adjacent(2, 3). adjacent(2, 5).
adjacent(3, 5). adjacent(3, 4). adjacent(4, 5). adjacent(5, 6).

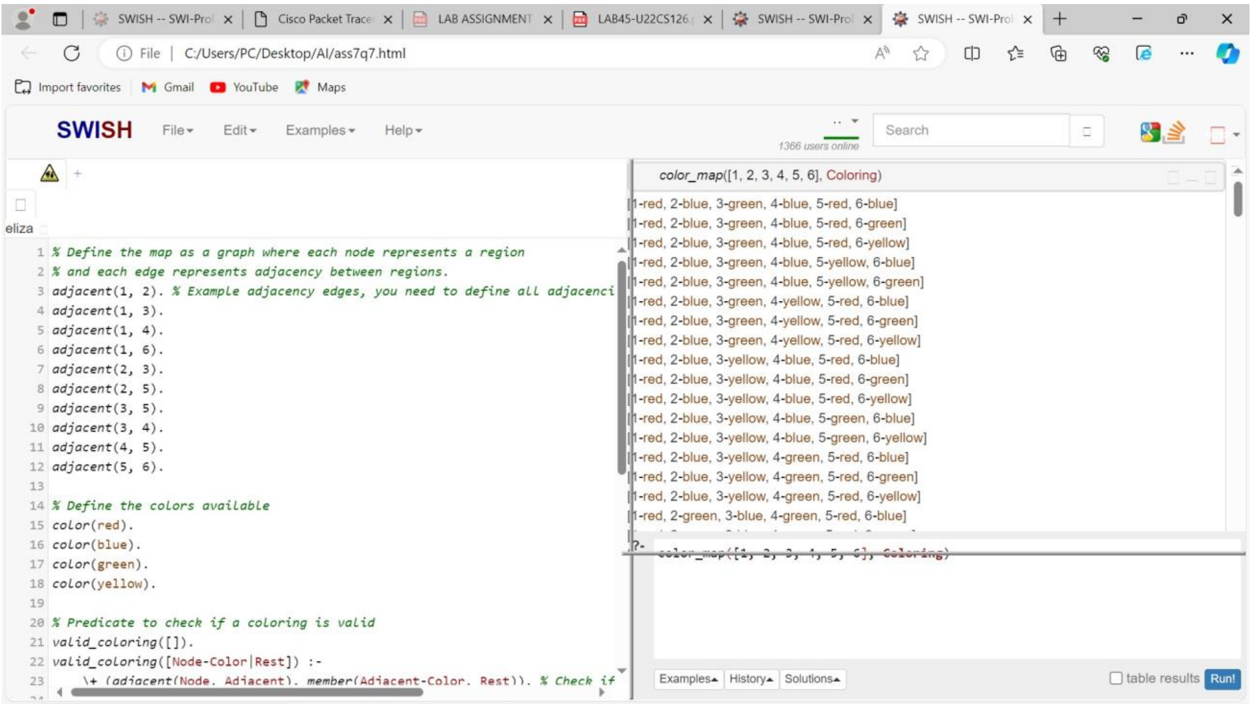
% Define the colors
available color(red).
color(blue). color(green).
color(yellow).

% Predicate to check if a coloring is valid
valid_coloring([]).
valid_coloring([Node-Color|Rest]) :-
\+ (adjacent(Node, Adjacent), member(Adjacent-Color, Rest)), % Check if adjacent nodes have the same color valid_coloring(Rest).

% Predicate to color the map
color_map(Map, Coloring) :-
color_nodes(Map, [], Coloring).

% Predicate to color the nodes recursively
color_nodes([], Coloring, Coloring).
color_nodes([Node|Nodes], PartialColoring, Coloring) :-
color(Color),
\+ member(Node-Color, PartialColoring), % Ensure no duplicate colors
append(PartialColoring, [Node-Color], UpdatedColoring), valid_coloring(UpdatedColoring), % Check if the updated coloring is valid color_nodes(Nodes, UpdatedColoring, Coloring).

% Query to find all possible colorings of the map
?- color_map([1, 2, 3, 4, 5, 6], Coloring), write(Coloring), nl, fail.


SWISH -- SWI-Prol
Cisco Packet Trace
LAB ASSIGNMENT
LAB45-U22CS126.
SWISH -- SWI-Prol
SWISH -- SWI-Prol
File | C:/Users/PC/Desktop/AI/ass7q7.html
Import favorites
Gmail
YouTube
Maps
SWISH
File
Edit
Examples
Help
1366 users online
Search
eliza
% Define the map as a graph where each node represents a region
% and each edge represents adjacency between regions.
adjacent(1, 2). % Example adjacency edges, you need to define all adjacenci
adjacent(1, 3).
adjacent(1, 4).
adjacent(1, 6).
adjacent(2, 3).
adjacent(2, 5).
adjacent(3, 5).
adjacent(3, 4).
adjacent(4, 5).
adjacent(5, 6).
% Define the colors available
color(red).
color(blue).
color(green).
color(yellow).
% Predicate to check if a coloring is valid
valid_coloring([]).
valid_coloring([Node-Color|Rest]) :-
\+ (adjacent(Node, Adjacent), member(Adjacent-Color, Rest)). % Check if
color_map([1, 2, 3, 4, 5, 6], Coloring)
[1-red, 2-blue, 3-green, 4-blue, 5-red, 6-blue]
[1-red, 2-blue, 3-green, 4-blue, 5-red, 6-green]
[1-red, 2-blue, 3-green, 4-blue, 5-red, 6-yellow]
[1-red, 2-blue, 3-green, 4-blue, 5-yellow, 6-blue]
[1-red, 2-blue, 3-green, 4-blue, 5-yellow, 6-green]
[1-red, 2-blue, 3-green, 4-yellow, 5-red, 6-blue]
[1-red, 2-blue, 3-green, 4-yellow, 5-red, 6-green]
[1-red, 2-blue, 3-green, 4-yellow, 5-red, 6-yellow]
[1-red, 2-blue, 3-yellow, 4-blue, 5-red, 6-blue]
[1-red, 2-blue, 3-yellow, 4-blue, 5-red, 6-green]
[1-red, 2-blue, 3-yellow, 4-blue, 5-red, 6-yellow]
[1-red, 2-blue, 3-yellow, 4-blue, 5-green, 6-blue]
[1-red, 2-blue, 3-yellow, 4-blue, 5-green, 6-yellow]
[1-red, 2-blue, 3-yellow, 4-green, 5-red, 6-blue]
[1-red, 2-blue, 3-yellow, 4-green, 5-red, 6-green]
[1-red, 2-blue, 3-yellow, 4-green, 5-red, 6-yellow]
[1-red, 2-green, 3-blue, 4-green, 5-red, 6-blue]
?- color_map([1, 2, 3, 4, 5, 6], Coloring)
Examples
History
Solutions
table results
Run!