Natural Language Processing
CSE 341

HW#4

Ömer Faruk Akduman
1801042094

Content

Homework Concept

In this Homework we are using the logical programming language that are prolog to implement some assignments.

1. Part-1

In here program can ability like these conditions:

• Check whether there is any scheduling conflict.

• Check which room can be assigned to a given class.

• Check which room can be assigned to which classes.

• Check whether a student can be enrolled to a given class.

• Check which classes a student can be assigned.

```
73
74  % Adds a student to the system with a given ID, list of courses, and handicapped status
75  add_student(ID, Courses, Handicapped) :-
76      assertz(student(ID, Courses, Handicapped)).
77
78  % Adds a course to the system with a given ID, instructor, capacity, hours, room ID, list of student IDs,
79  add_course(ID, Instructor, Capacity, Hours, RoomID, StudentIDs, Equipment, Handicapped) :-
80      assertz(course(ID, Instructor, Capacity, Hours, RoomID, StudentIDs, Equipment, Handicapped)).
81
82  % Adds a room to the system with a given ID, capacity, hours, equipment, and handicapped status
83  add_room(ID, Capacity, Hours, Equipment, Handicapped) :-
84      assertz(room(ID, Capacity, Hours, Equipment, Handicapped)).
85
86  % Checks if the schedules of two courses conflict
87  is_conflict(CourseID1, CourseID2) :-
88      % Find the hours of the two courses
89      doluluk(_, CourseID1, Hours1),
90      doluluk(_, CourseID2, Hours2),
91
92      % Check if there are any common elements in the lists of hours
93      common_ones(Hours1, Hours2).
94
95  % Returns true if there are any common elements in two lists, false otherwise
96  common_ones(List1, List2) :-
97      % Check if any element in List1 is a member of List2
98      member(Element, List1),
99      member(Element, List2).
00
```

```
is_conflict(CourseID1, CourseID2) :-
    % Find the hours of the two courses
    doluluk(_, CourseID1, Hours1),
    doluluk(_, CourseID2, Hours2),

    % Check if there are any common elements in the lists of hours
    common_ones(Hours1, Hours2).

% Returns true if there are any common elements in two lists, false otherwise
common_ones(List1, List2) :-
    % Check if any element in List1 is a member of List2
    member(Element, List1),
    member(Element, List2).

% Checks if a student can be enrolled in a course
check_enroll(StudentID, CourseID) :-
    % Check if the student is not already enrolled in the course
    \+ member(StudentID, StudentIDs),

    % Check if there is enough capacity in the course for the student
    length(StudentIDs, NumStudents),
    NumStudents < Capacity,

    % Check if the course has access for handicapped students (if the student is handicapped)
    (Handicapped = handicapped -> room(RoomID, _, _, _, Handicapped); true),

    % Check if the student is not enrolled in any courses that conflict with the given course
    findall(Conflict, (member(Course, Courses), is_conflict(CourseID, Course)), Conflicts),
    length(Conflicts, NumConflicts),
    NumConflicts = 0.
```

And also, we can check is there are conflict or not.

2- Part 2

Test 1

```
?- connection(ankara, X, C).
X = izmir,
C = 6 ;
X = diyarbakir,
C = 8 ;
X = rize,
C = 5 ;
X = istanbul,
C = 1 ;
X = van,
C = 4 ;
X = ankara,
C = 0 ;
X = antalya,
C = 8 ;
X = istanbul,
C = 8 ;
X = ankara,
C = 12 ;
X = izmir,
C = 6 ;
X = erzincan,
C = 11 ;
X = izmir,
C = 10
```

Test 2

```
?- connection(ankara, izmir, C).
C = 6 ;
```

Test 3

```
?- connection(canakkale, X , C).
X = erzincan,
C = 6 ;
```

Sourcode

```
%knowledge base
% schedule connections and its costs
schedules([schedule(canakkale, erzincan, 6),
        schedule(erzincan, canakkale, 6),
        schedule(erzincan, antalya, 3),
        schedule(antalya, erzincan, 3),
        schedule(izmir, antalya, 2),
        schedule(antalya, izmir, 2),
        schedule(diyarbakir, antalya, 4),
        schedule(antalya, diyarbakir, 4),
        schedule(izmir, istanbul, 2),
        schedule(istanbul, izmir, 2),
        schedule(rize, istanbul, 4),
        schedule(istanbul, rize, 4),
        schedule(izmir, ankara, 6),
        schedule(ankara, izmir, 6),
        schedule(diyarbakir, ankara, 8),
        schedule(ankara, diyarbakir, 8),
        schedule(rize, ankara, 5),
        schedule(ankara, rize, 5),
        schedule(ankara, istanbul, 1),
        schedule(istanbul, ankara, 1),
        schedule(ankara, van, 4),
        schedule(van, ankara, 4),
        schedule(gaziantep, van, 3),
        schedule(van, gaziantep, 3)]).
```

```
% Find the cost of a direct schedule between two cities
cost(X, Y, C) :- schedules(SCHEDULES), member(schedule(X, Y, C), SCHEDULES).

% Find the cost of the shortest route between two cities
connection(X, Y, C) :- connection(X, Y, C, []).

connection(X, Y, C, _) :- cost(X, Y, C).

% Base case: X and Z are the same node
connection(X, Z, C, Visited_place) :-
  X = Z,
  C = 0.

% Recursive case: X and Z are different nodes
connection(X, Z, C, Visited_place) :-
  % X is not in the list of visited nodes
  \+ member(X, Visited_place),

  % Find a neighboring node Y of X with a cost of CA
  cost(X, Y, CA),

  % Recursively find a connection from Y to Z with a cost of CB
  connection(Y, Z, CB, [X | Visited_place]),

  % The total cost is the sum of CA and CB
  C is CA + CB.
```

Resources

1- Stackoverflow
2- Wikipedia
3- https://metacpan.org/dist/AI-Prolog/view/lib/AI/Prolog/Cookbook.pod
4- https://www.seatavern.co.za/
5- https://gerrit-review.googlesource.com/Documentation/prolog-cookbook.html
6- ChatGPT