Gebze Technical University
Computer Engineering

CSE 341 - 2019 Fall

ASSIGNMENT 4 REPORT

FATİH KOÇ
141044013

Course Teacher: Yakup Genç

1 PART 1 – FLIGHT ROUTES

1.1 Problem Definition

Write the predicate “route(X,Y) – a route between X and Y exists” that returns true of if there is a route between any given two cities. A single query to complete your program should check if there is a direct route between two given cities. Alternatively, it can list all the connected cities for a given city.

1.2 Solution Approach and Implementation

To find there is a route or not between two cities, first will check is there any path between these two cities directly, then look for indirect connections by travelling graph.

```
%% CSE341 HW4 - Fatih Koç - 141044013
%% Part 1

%% knowledge base
flight(istanbul, antalya).
flight(istanbul, izmir).
flight(istanbul, ankara).
flight(istanbul, gaziantep).
flight(istanbul, van).
flight(istanbul, rize).
flight(edirne, edremit).
flight(edremit, erzincan).
flight(izmir, isparta).
flight(isparta, burdur).
flight(antalya, gaziantep).
flight(antalya, konya).
flight(konya, ankara).
flight(ankara, van).
flight(rize, van).

%% rules
link(X, Y):-
    flight(X, Y).
link(X, Y)
    flight(Y, X).

% returns member, taken from the internet
get_member(X, [Head|Tail]) :-
    X = Head;
    get_member(X, Tail).

route(X, Y):-
    path_check(X, Y, [X]).

path_check(X, Y, Cities):-
    link(X, Y),
    not(get_member(Y, Cities)).
path_check(X, Y, Cities) :-
    link(X, Z),
    not(get_member(Z, Cities)),
    append([Z], Cities, UpdatedCities),
    path_check(Z, Y, UpdatedCities).
```

1.3 Test Results

```
?- route(istanbul, izmir).
true ;
false.

?- route(istanbul, edirne).
false.

?- route(istanbul, izmir).
true
% Break level 1
[1]  ?- route(istanbul, burdur).
true
% Break level 2
[2]  ?- route(istanbul, paris).
false.

[2]  ?- route(istanbul, X).
X = antalya ;
X = izmir ;
X = ankara ;
X = gaziantep ;
X = van ;
X = rize ;
X = gaziantep ;
X = konya ;
X = ankara ;
X = van ;
X = rize ;
X = isparta ;
X = burdur ;
X = van ;
X = konya ;
X = rize ;
X = antalya ;
X = gaziantep ;
X = antalya ;
X = konya ;
X = ankara ;
X = van ;
X = rize ;
X = ankara ;
X = rize ;
X = konya ;
X = antalya ;
X = gaziantep ;
X = van ;
X = ankara ;
X = konya ;
X = antalya ;
X = gaziantep ;
false.
```

2 PART 2 – SHORTEST FLIGHT ROUTE

2.1 Problem Definition

Continuing with the previous problem, you are asked to write a program that checks if a route exists between two cities and if so, provides the shortest route.

2.2 Solution Approach and Implementation

First, added distance values of flights to knowledge base, provided from https://www.distancecalculator.net. Then, like previous part, travelled around graph for looking direct and indirect routes, but this time also calculated the total distances.

```
%% CSE341 HW4 - Fatih Koç - 141044013
%% Part 2

%% knowledge base
distance(istanbul, antalya, 481).
distance(istanbul, izmir, 328).
distance(istanbul, ankara, 351).
distance(istanbul, gaziantep, 847).
distance(istanbul, van, 1262).
distance(istanbul, rize, 967).
distance(edirne, edremit, 251).
distance(edremit, erzincan, 992).
distance(izmir, isparta, 308).
distance(isparta, burdur, 24).
distance(antalya, gaziantep, 592).
distance(antalya, konya, 192).
distance(konya, ankara, 227).
distance(ankara, van, 920).
distance(rize, van, 373).

%% rules
link(X, Y, Distance):-
    distance(X, Y, Distance).
link(X, Y, Distance):-
    distance(Y, X, Distance).

% returns member, taken from the internet
get_member(X, [Head|Tail]) :-
    X = Head;
    get_member(X, Tail).

sroute(X, Y, Distance):-
    path_check(X, Y, Distance, [X]).

path_check(X, Y, Distance, Cities):-
    link(X, Y, Distance),
    not(get_member(Y, Cities)).
path_check(X, Y, Distance, Cities) :-
    link(X, Z, CurrentDistance),
    not(get_member(Z, Cities)),
    append([Z], Cities, UpdatedCities),
    path_check(Z, Y, NextDistance, UpdatedCities),
    Distance is CurrentDistance + NextDistance.
```

2.3 Test Results

```
?- sroute(istanbul, antalya, X).
X = 481 ;
false.

?- sroute(istanbul, van, X).
X = 1262 ;
false.

?- sroute(edremit, erzincan, X).
X = 992 ;
false.
```

3 PART 3 – CLASSES, COURSES, STUDENTS, TIMES

3.1 Problem Definition

Write the predicates “when(X,Y) – time of the course X is Y”, “where(X,Y) – place of the course X is Y”, and “enroll(X,Y) – student X is enrolled in course Y”. Define/write a predicate “schedule(S,P,T)” that associates a student to a place and time of class. Define/write another predicate “usage(P,T)” that gives the usage times of a classroom. See the example query and its result. Define/write another predicate “conflict(X,Y)” that gives true if X and Y conflicts due to classroom or time. Define/write another predicate “meet(X,Y)” that gives true if student X and student Y are present in the same classroom at the same time.

3.2 Solution Approach and Implementation

There are some confusing points on this questions, what is asked on the questions and what is given as an example is different, so I implemented both ways, by adding ‘2’ to predicts names.

```
%% CSE341 HW4 - Fatih Koç - 141044013
%% Part 3

%% knowledge base
class(102, 10, z23).
class(108, 12, z11).
class(341, 14, z06).
class(455, 16, 207).
class(452, 17, 207).

enrollment(a, 102).
enrollment(a, 108).
enrollment(b, 102).
enrollment(c, 108).
enrollment(d, 341).
enrollment(e, 455).


%% rules
when(Course, Time) :-
    class(Course, Time, Room).
where(Course, Room) :-
    class(Course, Time, Room).
enroll(Student, Course) :-
    enrollment(Student, Course).
```

```
%% 3.1
schedule(Student, Room, Time) :-
    enrollment(Student, X), class(X, Time, Room).

schedule2(Student, Course, Time) :-
    enrollment(Student, Course), class(Course, Time, Room).


%% 3.2
usage(Room, Time) :-
    class(X, Time, Room).

usage2(Room, Course) :-
    class(Course, Time, Room).


%% 3.3
conflict(X, Y) :-
    class(X, Time, K), class(Y, Time, L).
conflict(X, Y) :-
    class(X, K, Room), class(Y, L, Room).

conflict2(X, Y) :-
    class(X, Time, Room), class(Y, Time, Room).


%% 3.4
meet(X, Y) :-
    enrollment(X, Course), enrollment(Y, Course).
```

3.3 Test Results

```
?- when(102, X).
X = 10.

?- where(102, X).
X = z23.

?- enroll(a, X).
X = 102 ;
X = 108.

?- enroll(X, 102).
X = a ;
X = b.
```

```
?- schedule(a, P, T).
P = z23,
T = 10 ;
P = z11,
T = 12.

?- schedule2(a, P, T).
P = 102,
T = 10 ;
P = 108,
T = 12.

?- usage(207, T).
T = 16 ;
T = 17.

?- usage2(207, T).
T = 455 ;
T = 452.
```

```
[2]  ?- conflict(455, 452).
true.

[2]  ?- conflict2(455, 452).
false.

[2]  ?- meet(a, b).
true ;
false.

[2]  ?- meet(a, c).
true.

[2]  ?- meet(a, d).
false.

[2]  ?- meet(a, X).
X = a ;
X = b ;
X = a ;
X = c.
```

4 PART 4 – SET OPERATIONS

4.1 Problem Definition

Define a Prolog predicate “element(E,S)” that returns true if E is in S. Define a Prolog predicate “union(S1,S2,S3)” that returns true if S3 is the union of S1 and S2. Define a Prolog predicate “intersect(S1,S2,S3)” that returns true if S3 is the intersection of of S1 and S2. Define a Prolog predicate “equivalent(S1,S2)” that returns true if S1 and S2 are equivalent sets.

4.2 Solution Approach and Implementation

I couldn’t be sure that I can use built-in functions or not, so I implemented both ways, by adding ‘2’ to predicts names which uses built-in functions.

```
%% CSE341 HW4 - Fatih Koç - 141044013
%% Part 4

%% 4.1
%% element implementation
element(E, [E|CDR]).
element(E, [CADR|CDR]) :- element(E, CDR).

%% element implementation with built-in functions.
element2(E, S) :- member(E, S).


%% 4.2
%% union implementation
union([], [], []).
union(S1, [], S1).
union([], S2, S2).
union([CAR1|CDR1], S2, S3) :-
    element(CAR1, S2),
    !,
    union(CDR1, S2, S3).
union([CAR1|CDR1], S2, [CAR1|S3]) :-
    union(CDR1, S2, S3).

%% union implementation with built-in functions
union2(S1, S2, S3) :-
    union(S1, S2, S3).


%% 4.3
%% intersection implementation
intersect([], _, []).
intersect(_, [], []).
intersect([CAR1|CDR1], S2, RESULT) :-
    element(CAR1, S2),
    !,
    RESULT = [CAR1|S3],
    intersect(CDR1, S2, S3).
intersect([_|CDR1], S2, S3) :-
    intersect(CDR1, S2, S3).

%% intersection implementation with built-in functions
intersect2(S1, S2, S3) :- intersection(S1, S2, S3).

%% 4.4
%% equivalent implementation with recursion
equivalent(S1, S2) :- intersect(S1, S2, S1), intersect(S1, S2, S2).
```

4.3 Test Results

```
% Break level 4
[4]  ?- element(a, [a, b, c]).
true
% Break level 5
[5]  ?- element(a, [b, c]).
false.

[5]  ?- union([a, b], [b, c], [a, b, c]).
true.

[5]  ?- union([a, b], [], [a, b, c]).
false.

[5]  ?- intersect([a, b], [], [a, b, c]).
false.

[5]  ?- intersect([a, b], [], []).
true
% Break level 6
[6]  ?- intersect([a, b], [a], [a]).
true
% Break level 7
[7]  ?- equivalent([a, b, c], [a, b]).
false.

[7]  ?- equivalent([a, b, c], [a, b, c]).
true
```