CSE341 – Programming Languages (Fall 2019)
Homework #4

FURKAN ÖZEV 161044036

PART – 1:

Firstly, I created the possible flight graph given in the homework pdf as knowledge base.

```
flight(edirne,edremit).

flight(edremit,edirne).
flight(edremit,erzincan).

flight(erzincan,edremit).

flight(burdur,ısparta).

flight(ısparta,burdur).
flight(ısparta,izmir).

flight(izmir,ısparta).
flight(izmir,istanbul).

flight(istanbul,izmir).
flight(istanbul,antalya).
flight(istanbul,gaziantep).
flight(istanbul,ankara).
flight(istanbul,van).
flight(istanbul,rize).

flight(antalya,istanbul).
flight(antalya,konya).
flight(antalya,gaziantep).

flight(gaziantep,istanbul).
flight(gaziantep,antalya).

flight(konya,antalya).
flight(konya,ankara).

flight(ankara,konya).
flight(ankara,istanbul).
flight(ankara,van).

flight(van,ankara).
flight(van,istanbul).
flight(van,rize).

flight(rize,van).
flight(rize,istanbul).
```

flight(X,Y) is a fact that show that X and Y cities has a flight.

These are all fact of part1.

Secondly, I created predict that check whether is a direct route between two given cities. And this predict can list all the connected cities for a given city.

route(X,Y) is a predicate that indicate there exist a route between X and Y if there is flight between X and Y.

```
%rules..
route(X, Y) :- routex(X, Y, []).
routex(X, Y, VisitList) :- flight(X, Z), not(member(Z, VisitList))
                         , (Y = Z; routex(Z, Y, [X | VisitList])).
```

I search with an list for that predicate works properly avoiding infinite loops.

route(X,Y) predicate call helper predicate to search flights.

This predicate takes list as 3.input for avoid infinite loops.

This list stores the visited cities in order not to visit them again. This list name is VisitList.

1- flight(X,Z) → It takes a flight from X to another city (Z is destination city). If there is no flight, It will return false. Else continue for second step.

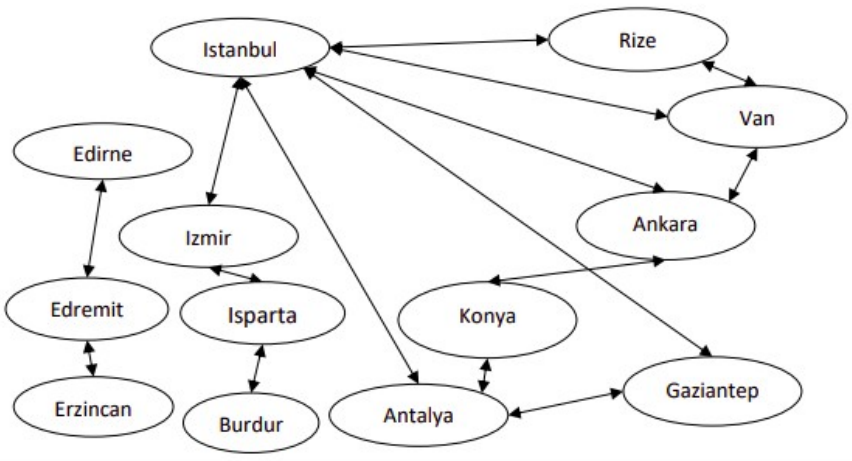
2- not(member(Z, VisitList)) → Check if the destination city has been visited. If destination city (Z) doesn't exist the VisitList, it has not been visited before. If it is visited, return false. Else continue for third step.

3- A direct flight from X to Y may exist or transfer must be made. If the destination city (Z) is same the destination city (Y) entered, So there are direct flight X to Y. Else, There is a transfer like X to Z then Z to Y.

4- Y = Z → Indicates that the received flight is a direct flight if the destination city (Z) is the same as the destination city (Y) entered. So, it indicates that there is a direct flight from X to Y.

5- OR routex(Z, Y, [X | VisitList]) → If there is a no direct flight, Check whether the transfer exists. This transfer must be Z to Y. X is added to the list because is visited.

6- If these conditions are met, there is flight. Thus, it is possible to list the cities that meet this condition in the query.



TEST:

Test for directed **flight: (You can check on above graph!)**

```
?- flight(istanbul,izmir).
true.

?- flight(ankara,konya).
true.

?- flight(gaziantep,ısparta).
false.

?- flight(edirne,edremit).
true.

?- flight(edirne,erzincan).
false.

?- flight(ankara,rize).
false.

?- flight(rize,van).
true.
```

There is no directed flight:

gaziantep → ısparta

edirne → erzincan

ankara → rize

There is a directed flight:

istanbul → izmir

ankara → konya

edirne → edremit

rize → van

Test for **route.: (You can check on above graph!)**

```
?- route(edirne, X).
X = edremit ;
X = erzincan .

?- route(edremit, edirne).
true .

?- route(erzincan, edirne).
true .

?- route(erzincan, edremit).
true .

?- route(edirne, ankara).
false.

?- route(edremit, ankara).
false.

?- route(erzincan, ankara).
false.
```

In this example you see the flights from Edirne.

And it really checks to see if a flight exists.

And there is no flight from Edirne to Ankara, Also there is no flight from cities that have flights from Edirne to Ankara.

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

In this example you see the flights from Istanbul.

In Predict, It visit all possible path, So it can write city more than 1. But there is no infinity loop. I prevented it with VisitList.

Here you can see all the destinations that you can go through all (direct or indirect) flights.

```
?- route(istanbul, burdur).
true .

?- route(istanbul, van).
true .

?- route(istanbul, edremit).
false.

?- route(istanbul, edirne).
false.
```

As you can see in the previous result, there are no flights from Istanbul to Edremit and Edirne. And there are flights from Istanbul to Burdur and Van.

PART – 2:

In the first step, I expanded the knowledge base by adding distances for the direct flights.

I calculated distances via given url link.

```
%facts..
distance(edirne,edremit,235).

distance(edremit,edirne,235).
distance(edremit,erzincan,1066).

distance(erzincan,edremit,1066).

distance(burdur,ısparta,25).

distance(ısparta,burdur,25).
distance(ısparta,izmir,309).

distance(izmir,ısparta,309).
distance(izmir,istanbul,329).

distance(istanbul,izmir,329).
distance(istanbul,antalya,483).
distance(istanbul,gaziantep,847).
distance(istanbul,ankara,352).
distance(istanbul,van,1262).
distance(istanbul,rize,373).

distance(antalya,istanbul,483).
distance(antalya,konya,192).
distance(antalya,gaziantep,592).

distance(gaziantep,istanbul,847).
distance(gaziantep,antalya,592).

distance(konya,antalya,192).
distance(konya,ankara,227).

distance(ankara,konya,227).
distance(ankara,istanbul,352).
distance(ankara,van,920).

distance(van,ankara,920).
distance(van,istanbul,1262).
distance(van,rize,373).

distance(rize,van,373).
distance(rize,istanbul,373).
```

distance(X,Y, C) is a fact that show that X and Y cities has a flight and the distance is C.

Secondly, I created predict that check whether is a direct route between two given cities and the shortest distance between them. And this predict return shortest distance between X to Y cities.

sroute(X,Y, C) is a predicate. If there is a route from X to Y, it returns the shortest route.

```
%rules..
sroute(C1,C2,X) :- flight(C1,C2), distance(C1,C2,X).
sroute(C1,C2,X) :- flight(C1,C3), flight(C3,C2), distance(C1,C3,X1)
                   , distance(C3,C2,X2), X is X1+X2.
```

1- If there is directed flight from City1 to City2, It will return distance between City1 and City2.

2- If there is no directed flight, Then I must check whether there is indirected(transfer) flight(or route).

3- flight(C1,C3), flight(C2,C3) → show that wheter there is indirected flight. If there are continue step 4. Else return false.

4- distance(C1,C3,X1) → Calculate distance between C1 and C3.

5- distance(C3, C2, X2) → Calculate distance between C3 and C2.

6- X is X1 + X2 → Calculate total distance between C1 and C2.

TEST:

Test for directed sroute: (You can check on above image!)

If there is a direct flight, it takes the distance directly from the flight. Otherwise it takes the distance over the transfer.

```
?- sroute(edremit,erzincan,X).
X = 1066 .

?- sroute(istanbul,izmir,X).
X = 329 .

?- sroute(konya,van,X).
X = 1147.
```

In this example you see the distance of shortest route (or path) between cities.

And it really checks to see if a route (or path) exists.

```
?- sroute(konya,edremit,X).
false.

?- sroute(edirne,istanbul,X).
false.

?- sroute(erzincan,gaziantep,X).
false.
```

And there is no flight from Konya to Edremit.

And there is no flight from Edirne to Istanbul.

And there is no flight from Erzincan to Gaziantep.

PART – 3:

In the first step, I created the knowledge base with given database about classes, classrooms and students in homework pdf.

Classes		
Class	Time	Room
102	10	z23
108	12	z11
341	14	z06
455	16	207
452	17	207

Enrollment	
Student	Class
a	102
a	108
b	102
c	108
d	341
e	455

```
%facts..
when(102, 10).
when(108, 12).
when(341, 14).
when(455, 16).
when(452, 17).

where(102, z23).
where(108, z11).
where(341, z06).
where(455, 207).
where(452, 207).

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

when(X,Y) is a fact that indicates that time of the course X is Y.

where(X,Y) is a fact that indicates that place of the course X is Y.

enroll(X,Y) is a fact that indicate that student X is enrolled in course Y.

3.1:

I created a predicate “schedule(S,P,T)” that associates a student to a place and time of class.

```
schedule(S, P, T) :- enroll(S, C), where(C, P), when(C,T).
```

schedule(S, P, T) → S is Given Student Name, P is Classroom, T is Time.

This predict takes the student name and returns the place and time of the courses the student takes.

enroll(S, C) → C becomes the classes (courses, not classroom) associated with the entered student.

where(C,P) → P becomes the place of the current course.

when(C,T) → T becomes the time of the current course.

TEST:

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

?- schedule(b,P,T).
P = z23,
T = 10.

?- schedule(c,P,T).
P = z11,
T = 12.

?- schedule(d,P,T).
P = z06,
T = 14.

?- schedule(e,P,T).
P = 207,
T = 16.

?- schedule(f,P,T).
false.
```

In this example you see schedule of students.

Returns false if an invalid student name is entered. So schedule(f,P,T) return false.

3.2:

I created a predicate “usage(P,T)” that gives the usage times of a classroom.

```
usage(P, T) :- where(C,P), when(C,T).
```

usage(P, T) → P is Classroom, T is Time.

This predict takes the classroom and return the usage time of classroom.

where(C, P) → C becomes the classes (courses, not classroom) associated with the entered place(classroom).

when(C,T) → T becomes the time of the current course.

TEST:

```
?- usage(z23,T).
T = 10.

?- usage(z11,T).
T = 12.

?- usage(z06,T).
T = 14.

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

?- usage(209,T).
false.
```

In this example you see usage times of classrooms.

Returns false if an invalid classroom is entered. So usage(209,T) return false.

3.3:

I created a predicate “conflict(X,Y)” that gives true if X and Y conflicts due to classroom or time.

```
conflict(X,Y):- (when(X,T1), when(Y,T2), T1==T2); (where(X,P1), where(Y,P2), P1==P2).
```

conflict(X, Y) → X and Y are course(class).

This predict takes the course(class) then if there is conflict return true, else return false.

when(X, T1) → T1 becomes the time of X course.

when(Y, T2) → T2 becomes the time of Y course.

T1==T2 → If T1 and T2 is same time, so there is a conflict due to time, then return true.

OR

where(X, P1) → P1 becomes the classrrom associated with the X course(class).

where(Y, P2) → P2 becomes the classrrom associated with the Y course(class).

P1==P2 → If P1 and P2 is same class, so there is a conflict due to place, then return true.

TEST:

```
?- conflict(455,452).
true.

?- conflict(455,102).
false.
```

In this example you see that there is a conflict between 455 and 452 courses due to time, And there is no conflict between 405 and 102 courses.

3.4:

I created a predicate “conflict(X,Y)” that gives true if student X and student Y are present in the same classroom at the same time.

```
meet(X,Y):- enroll(X,C1), enroll(Y,C2), C1==C2,!.
```

conflict(X, Y) → X and Y are student.

This predict takes the student then if students take same course(class) return true, else return false.

enroll(X, C1) → C1 becomes the classes (courses, not classroom) associated with the student X.

enroll(Y, C2) → C2 becomes the classes (courses, not classroom) associated with the student Y.

C1==C2 → If C1 and C2 is same course(class), so students will present in the same classroom at same time, then return true. Else return false.

TEST:

```
?- meet(a,b).
true.

?- meet(a,c).
true.

?- meet(a,d).
false.

?- meet(a,e).
false.

?- meet(d,e).
false.
```

In this example:

a and b will meet. Because Student A take course 102 and Student B take course 102.

a and c will meet. Because Student A take course 108 and Student C take course 108.

a and d won't meet. Because No common courses.

a and e won't meet. Because No common courses.

d and e won't meet. Because No common courses.

PART – 4:

4.1:

predicate "element(E,S)" that returns true if E is in S.

E is element, and S is list.

```
element(E, [E|_]).
element(E, [_|S]):- element(E, S).
```

element(E, [E|_]) → If element exist in head of list, return true.

element[E, [_|S]) :- element(E,S) → Recursive call with tail of list.

TEST:

```
?- element(5, [1,5,7,9,4]).
true .

?- element(7, [1,5,7,9,4]).
true .

?- element(11, [1,5,7,9,4]).
false.
```

In this example 5 and 11 exist in list and 11 doesn't exist in list.

```
?- element(X, [1,5,7,9,4]).
X = 1 ;
X = 5 ;
X = 7 ;
X = 9 ;
X = 4 ;
false.
```

In this example you see all element of list.

4.2:

predicate “union(S1,S2,S3)” that returns true if S3 is the union of S1 and S2.

S1, S2 and S3 are list.

```
union(S1,S2,S3) :- union2(S1,S2,X), equivalent(X,S3).
union2([], S2, S2).
union2([E|S1], S2, S3) :- element(E, S2), !, union2(S1, S2, S3).
union2([E|S1], S2, [E|S3]) :- union2(S1, S2, S3).
```

union(S1,S2,S3) :- union2(S1,S2,X), equivalent(X,S3). → This is to assume that the predict union set can be in random order.

union2(S1,S2,X) → X will be union of S1,S2.

equivalent(X,S3) → Check wheter union result X is equivalent entered union S3.

If they are equivalent then return true, else return false.

TEST:

```
?- union([3,5,7,6],[4,5,7,8,3],[3,5,7,6,4,8]).
true .

?- union([3,5,7,6],[4,5,7,8,3],[4,3,7,5,8,6]).
true .

?- union([3,5,7,6],[4,5,7,8,3],[4,3,7,5,8]).
false.

?- union([3,5,7,6],[4,5,7,8,3],[4,3,7,5,8,6,11]).
false.
```

The union set entered in the first two examples is correct. But order is different.

The union set in third example doesn’t exist 6, so it is false.

The union set in fourth example exists different value, so it is false.

4.3:

predicate “intersect(S1,S2,S3)” that returns true if S3 is the intersection of of S1 and S2.

```
intersect(S1,S2,S3) :- intersect2(S1,S2,X), equivalent(X,S3).
intersect2([], _, []).
intersect2([E|S1], S2, [E|S3]):- element(E, S2), !, intersect2(S1, S2, S3).
intersect2([_|S1], S2, S3):- intersect2(S1, S2, S3).
```

```
?- intersect([3,5,7,6], [4,5,7,8,3], [3,5,7]).
true .

?- intersect([3,5,7,6], [4,5,7,8,3], [7,5,3]).
true .

?- intersect([3,5,7,6], [4,5,7,8,3], [7,5]).
false.

?- intersect([3,5,7,6], [4,5,7,8,3], [7,5,3,6]).
false.
```

4.4:

predicate “equivalent(S1,S2)” that returns true if S1 and S2 are equivalent sets.

```
equivalent(S1, S2) :- equivalent2(S1,S2), equivalent2(S2,S1).
equivalent2([],_).
equivalent2([E|S1],S2):- element(E,S2), equivalent2(S1,S2).
```

```
?- equivalent([4,8,11,9,5], [4,8,11,9,5]).
true .

?- equivalent([5,8,9,11,4], [11,4,5,9,8]).
true .

?- equivalent([5,8,9,12,4], [11,4,5,9,8]).
false.

?- equivalent([5,8,9,12,4], [4,5,9,12]).
false.

?- equivalent([5,8,9,12,4], [4,5,9,12,8,13]).
false.
```

I couldn't explain enough 4th part in the report, because I don't have time. But I designed predicates as clearly as possible.