CSE 341 – PROGRAMMING LANGUAGES

HW4 – REPORT

Elif Goral

171044003

Part 1:

Firstly I define the facts which are flights between the cities.

```
flight(edirne,edremit).
flight(edremit,edirne).
flight(edremit,erzincan).
flight(erzincan,edremit).
flight(istanbul,izmir).
flight(izmir,istanbul).
flight(istanbul,antalya).
flight(antalya,istanbul).
flight(istanbul,gaziantep).
flight(gaziantep,istanbul).
flight(istanbul,ankara).
flight(ankara,istanbul).
flight(istanbul,van).
flight(van,istanbul).
flight(istanbul,rize).
flight(rize,istanbul).
flight(burdur,ısparta).
flight(ısparta,burdur).
flight(ısparta,izmir).
flight(izmir,ısparta).
flight(antalya,konya).
flight(konya,antalya).
flight(antalya,gaziantep).
flight(gaziantep,antalya).
flight(konya,ankara).
flight(ankara,konya).
flight(ankara,van).
flight(van,ankara).
flight(van,rize).
flight(rize,van).
```

That rule tries all the fact of flight. If fight from start city to intermediate city can ocur and that city is not in VisitList, searching is continue. Recursive searching happens starting from start city. For example, if searching starts in A city and customer can fight from A to B and C cities, The client goes from city a to city b and looks at the places she/he can go from there. Then, client goes from city a to city c and looks at the places she/he can go from there. This is how a recursive call search the cities. Because of the recursive call, some cities shown more than once.

```
route(Start, End)              :-    helper(Start, End, []).

helper(Start, End, VisitList) :-    flight(Start, Intermediate),
                                    not(member(Intermediate, VisitList)),
                                    (End = Intermediate; helper(Intermediate, End, [Start | VisitList])).
```

Test results:

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

?- route(istanbul,edirne).
false.

?- route(istanbul,erzincan).
false.

?- route(istanbul,burdur).
true .

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

```
?- 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 ;
false.
```

```
?- route(izmir,X).
X = istanbul ;
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 ;
X = isparta ;
X = burdur ;
false.
```

Part 2:

```
flight(edirne,edremit).
flight(edremit,edirne).
flight(edremit,erzincan).
flight(erzincan,edremit).
flight(istanbul,izmir).
flight(izmir,istanbul).
flight(istanbul,antalya).
flight(antalya,istanbul).
flight(istanbul,gaziantep).
flight(gaziantep,istanbul).
flight(istanbul,ankara).
flight(ankara,istanbul).
flight(istanbul,van).
flight(van,istanbul).
flight(istanbul,rize).
flight(rize,istanbul).
flight(burdur,ısparta).
flight(ısparta,burdur).
flight(ısparta,izmir).
flight(izmir,ısparta).
flight(antalya,konya).
flight(konya,antalya).
flight(antalya,gaziantep).
flight(gaziantep,antalya).
flight(konya,ankara).
flight(ankara,konya).
flight(ankara,van).
flight(van,ankara).
flight(van,rize).
flight(rize,van).
```

```
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(antalya,istanbul,483).
distance(istanbul,antalya,483).
distance(istanbul,gaziantep,847).
distance(gaziantep,istanbul,847).
distance(ankara,istanbul,352).
distance(istanbul,ankara,352).
distance(istanbul,van,1262).
distance(van,istanbul,1262).
distance(rize,istanbul,968).
distance(istanbul,rize,968).
distance(konya,antalya,192).
distance(antalya,konya,192).
distance(antalya,gaziantep,592).
distance(gaziantep,antalya,592).
distance(konya,ankara,227).
distance(ankara,konya,227).
distance(ankara,van,920).
distance(van,ankara,920).
distance(van,rize,373).
distance(rize,van,373).
```

Firstly I control the direct fight from start city to end city. Then calculate the distance.

```
sroute(StartCity,EndCity,Distance) :-   flight(StartCity,EndCity),
                                        distance(StartCity,EndCity,Distance).
```

I control the route which has one extra flight in between start city and end city.

```
sroute(StartCity,EndCity,Distance) :-   flight(StartCity,IntermediateCity),
                                        flight(IntermediateCity,EndCity),
                                        distance(StartCity,IntermediateCity,Distance1),
                                        distance(IntermediateCity,EndCity,Distance2),
                                        Distance is Distance1+Distance2.
```

I control the route which has two extra flight in between start city and end city. I did not check any more extra flights. I checked the situation of two flights between Istanbul and Burdur. No more extra flights is needed on flights between other cities.

```
sroute(StartCity,EndCity,Distance) :-   flight(StartCity,IntermediateCity),
                                        flight(IntermediateCity,IntermediateCity2),
                                        flight(IntermediateCity2,EndCity),
                                        distance(StartCity,IntermediateCity,Distance1),
                                        distance(IntermediateCity,IntermediateCity2,Distance2),
                                        distance(IntermediateCity2,EndCity,Distance3),
                                        Distance is Distance1+Distance2+Distance3.
```

Test results:

```
?- sroute(izmir,istanbul,X).
X = 329 ;
X = 987 ;
X = 1295 ;
X = 2023 ;
X = 1033 ;
X = 2853 ;
X = 2265 ;
X = 947.

?- sroute(izmir,ankara,X).
X = 681 ;
X = 2511 ;
false.

?- sroute(istanbul,burdur,X).
X = 663 ;
false.
```

Part 3:

```
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).
```

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

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

3.2) predicate “usage(P,T)” that gives the usage times of a classroom. See the example query and its result.

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

3.3) Predicate “conflict(X,Y)” that gives true if X and Y conflicts due to classroom or time. X and Y are classNames:

```
conflict(X,Y) :- (when(X,Time1), when(Y,Time2), Time1==Time2) ;
                 (where(X,Place1), where(Y,Place2), Place1==Place2).
```

3.4) X and Y are students. Predicate “meet(X,Y)” that gives true. if student X and student Y are present in the same classroom at the same time. For this purpose, I control the same classes. If students are in same class which means They are in same classroom at the same time.

```
meet(X,Y):-     enroll(X,ClassName1),
                enroll(Y,ClassName2),
                ClassName1==ClassName2, !.
```

Test Results:

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

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

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

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

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

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

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

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

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

```
?- conflict(102,108).
false.

?- conflict(102,341).
false.

?- conflict(102,455).
false.

?- conflict(102,452).
false.

?- conflict(108,341).
false.

?- conflict(108,455).
false.

?- conflict(108,452).
false.

?- conflict(341,452).
false.

?- conflict(341,455).
false.

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

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

?- meet(a,c).
true.

?- meet(a,d).
false.

?- meet(a,e).
false.

?- meet(b,c).
false.

?- meet(b,d).
false.

?- meet(b,e).
false.

?- meet(c,d).
false.

?- meet(c,e).
false.

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

Part 4 :

4.1) predicate “element(E,S)” that returns true if E is in S. Firstly I control the position of element. If element exist in head of list, It returns true, otherwise it continues to next statement. Which looks rest of the list with recursive calls

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

4.2) Predicate “union(S1,S2,S3)” that returns true if S3 is the union of S1 and S2. Firstly I find the union of S1 and S2, Then I control the union and S3 are equivalent or not.

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

4.3) Predicate “intersect(S1,S2,S3)” that returns true if S3 is the intersection of of S1 and S2. Firstly I find the intersection of S1 and S2, Then I control the intersection and S3 are equivalent or not.

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

4.4) Predicate “equivalent(S1,S2)” that returns true if S1 and S2 are equivalent sets. If s1 and s2 lists are empty, returns true. otherwise, I control the elements of s1 list in s2 list respectively. Then I do same process for s2 list. I control the elements of s2 list in s1 list respectively.

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

Test Results:

```
?- element(1,[1,5,6]).
true .

?- element(1,[2,5,6]).
false.

?- element(5,[2,5,6]).
true .

?- element(5,[2,7,6]).
false.
```

```
?- union([1,2,5],[7,8,3],[1,5,7,8,3,2]).
true .

?- union([1,2,5],[7,8,3],[1,5,7,8,2]).
false.

?- union([1,2,5],[7,8,3],[1,5,7,8,2,4]).
false.
```

```
?- intersect([1,2,5],[8,2,1],[1,2]).
true .

?- intersect([1,2,5],[8,2,1],[1]).
false.

?- intersect([1,2,5],[8,2,1],[1,8]).
false.

?- intersect([8,2,7],[9,3,5],[2,3]).
false.

?- intersect([8,2,7],[9,3,5],[]).
true.
```

```
?- equivalent([8,2,7],[9,3,5]).
false.

?- equivalent([8,2,7],[8,2,7]).
true .

?- equivalent([8,2,7],[8,2]).
false.
```