COMP 3031 Assignment 3

Logic Programming

Fall 2021

Due: 5PM on Nov 26 Friday

Instructions

- There are five problems in this assignment. Each problem counts for two points.
- Write your prolog program according to the definition of the problem, with the same predicate name and number of arguments as specified. Write all the solutions in a single file named "ass3.pl". You can use any helper predicates, including built-in predicates available on the lab computers, but *not* any external modules that require downloading.
- Submit your code through Canvas.
- **No late submissions will be accepted.**
- Your submission will be loaded into PROLOG on a lab 2 machine with the following command:
  "?- [ass3].".
  If your submission cannot be loaded into PROLOG, no points will be given.

For the five problems in this assignment, we define a relation `flight(Src,Dst)` representing a direct flight from a city `Src` to another city `Dst`. Assume city `Src` and city `Dst` are different. For example, the fact `flight(0,1)` represents a direct flight from city `0` to city `1`.
You can assume all input are valid in the PROLOG interpreter.

Examples are based on the following database:

```
/* The database of flight facts */
flight(0, 1).
flight(3, 4).
flight(1, 2).
flight(2, 0).
flight(2, 1).
flight(3, 2).
flight(4, 3).
```

Question 1.

Define a relation `reachable(SrcList, Dst)` in which all cities in SrcList can reach city `Dst` via a direct flight or connecting flights. The city IDs in `SrcList` are unique and different from `Dst`, but the order of city IDs in `SrcList` is unimportant.

Examples:

```
?- reachable([], 1).
false.

?- reachable([1], 2).
true ;
false.

?- reachable(X, 0).
X = [1] ;
X = [1, 2] ;
X = [1, 2, 3] ;
X = [1, 2, 3, 4] ;
X = [1, 2, 4] ;
X = [1, 3] ;
X = [1, 3, 4] ;
X = [1, 4] ;
X = [2] ;
X = [2, 3] ;
X = [2, 3, 4] ;
X = [2, 4] ;
X = [3] ;
X = [3, 4] ;
X = [4] ;
false.

?- reachable([2,3], X).
X = 0 ;
X = 1 ;
false.

?- reachable([5], Dst).
false.
```

Question 2.

Define a relation `all_cities(L, N)` that specifies a list `L` of `N` unique cities in the order the cities appear in the flight facts. `N` is a number less than or equal to the total number of the unique cities appearing in the flight facts.

Examples:

```
?- all_cities(L, 0).
L = [].

?- all_cities(L, 1).
L = [0].

?- all_cities(L, 2).
L = [0, 1].

?- all_cities(L, 3).
L = [0, 1, 3].

?- all_cities(L, 5).
L = [0, 1, 3, 4, 2].

?- all_cities([0, 1, 3], 3).
true.

?- all_cities([0, 1, 3], 5).
false.

?- all_cities([4, 1, 0], 3).
false.
```

Question 3.

Write a relation `count_paths(Src, Dst, N)` in which `N` is the number of paths (each path contains a single flight or a sequence of connecting flights) from city `Src` to city `Dst`. All cities in each counted path must be unique, i.e., no city appears more than once in the path.

Examples:

```
?- count_paths(0, 1, N).
N = 1.

?- count_paths(0, 2, N).
N = 1.
```

```
?- count_paths(2, 1, N).
N = 2.

?- count_paths(X, Y, 2).
X = 2,
Y = 1 ;
X = 3,
Y = 1 ;
X = 4,
Y = 1 ;
false.

?- count_paths(X, X, N).
false.

?- count_paths(X, 2, N).
X = 0,
N = 1 ;
X = N, N = 1 ;
X = 3,
N = 1 ;
X = 4,
N = 1.

?- count_paths(2, X, N).
X = 0,
N = 1 ;
X = 1,
N = 2.

?- count_paths(X, Y, N).
X = 0,
Y = N, N = 1 ;
X = 0,
Y = 2,
N = 1 ;
X = N, N = 1,
Y = 0 ;
X = N, N = 1,
Y = 2 ;
X = 2,
Y = 0,
N = 1 ;
```

```
X = N, N = 2,
Y = 1 ;
X = 3,
Y = 0,
N = 1 ;
X = 3,
Y = 1,
N = 2 ;
X = 3,
Y = 2,
N = 1 ;
X = 3,
Y = 4,
N = 1 ;
X = 4,
Y = 0,
N = 1 ;
X = 4,
Y = 1,
N = 2 ;
X = 4,
Y = 2,
N = 1 ;
X = 4,
Y = 3,
N = 1.
```

Question 4.

Write a relation `shortest_paths(Src, Dst, L)` that specifies a list `L` containing all the shortest paths from city `Src` to city `Dst`. Assume Src and Dst are different. If more than one path is the shortest, all these paths are included in `L`. If there is no path from city `Src` to city `Dst`, `L` is an empty list. The paths in `L` are unique, but the order these paths appear in `L` is unimportant.

Examples:

```
?- shortest_paths(0, 1, L).
L = [[0, 1]].

?- shortest_paths(4, 2, L).
L = [[4, 3, 2]].

?- shortest_paths(2, 1, L).
L = [[2, 1]].
```

```
?- shortest_paths(0, 1, [[0, 1]]).
true.

?- shortest_paths(2, 1, [[2, 0, 1]]).
false.

?- shortest_paths(X,Y,Z).
X = 0,
Y = 1,
Z = [[0, 1]] ;
X = 0,
Y = 2,
Z = [[0, 1, 2]] ;
X = 1,
Y = 0,
Z = [[1, 2, 0]] ;
X = 1,
Y = 2,
Z = [[1, 2]] ;
X = 2,
Y = 0,
Z = [[2, 0]] ;
X = 2,
Y = 1,
Z = [[2, 1]] ;
X = 3,
Y = 0,
Z = [[3, 2, 0]] ;
X = 3,
Y = 1,
Z = [[3, 2, 1]] ;
X = 3,
Y = 2,
Z = [[3, 2]] ;
X = 3,
Y = 4,
Z = [[3, 4]] ;
X = 4,
Y = 0,
Z = [[4, 3, 2, 0]] ;
X = 4,
Y = 1,
Z = [[4, 3, 2, 1]] ;
X = 4,
```

```
Y = 2,
Z = [[4, 3, 2]] ;
X = 4,
Y = 3,
Z = [[4, 3]].
```

Question 5.

Write a relation `search_destination(L, Len, Src)` that specifies a list `L` of cities each of which is reachable from the city `Src` within the given path length `Len`. Assume Src and Dst are different. The city IDs in `L` are unique and different from Src, but the order of city IDs in `L` is unimportant.

Examples:

```
?- search_destination(L, 0, 0).
L = [].

?- search_destination(L, 1, 0).
L = [1].

?- search_destination(L, 2, 0).
L = [1, 2].

?- search_destination([1], 1, 0).
true.

?- search_destination([0, 1], 2, 0).
false.

?- search_destination([1], 1, Src).
Src = 0 ;
false.
```