PROJECT 3

Course: CSE 537

Due date: October 31st, 11:59pm

TEAM MEMBER 1:
Name: Ayushi Srivastava
Student Id: 112101239
Email: ayushi.srivastava@stonybrook.edu

TEAM MEMBER 2:
Name: Jahnavi Tharigopula
Student Id: 112078393
Email: jahnavi.tharigopula@stonybrook.edu

The input is defined as follows:

```
% INPUT
% Defined 4 Users, 5 Roles and 5 Permissions
users(5).
roles(5).
perms(5).

% Roles for each User
% User 1 ,2, 3 and 4 have single or multiple direct roles defined
ur(1,2).
ur(2,3).
ur(3,4).
ur(4,4).
ur(2,4).

% Permissions for each Role
% Roles 3, 4 have No permissions, whereas 2 has multiple permissions
rp(1,2).
rp(2,4).
rp(3,5).
rp(4,2).
rp(5,2).

% Role Hierarchy
% There exists a circular Hierarchy between 4->5->4 and 3->5->3.
rh(1,3).
rh(4,5).
rh(5,4).
rh(5,3).
rh(2,3).
rh(3,5).
```

Question 1:

For Question 1, we are required to return all the authorized roles for a given User.

Cases Considered:

- User may not have any role defined.
    - Function should return that nothing, meaning that User has no roles defined.

        - User 5 has no roles defined, hence the function returns an empty list.

            -
              ```
              authorized_roles(5, Y).
              Y = [];
              no
              ```

- User may have '*single or multiple direct roles'* defined but there is no hierarchy from these roles to any other roles.
    - Function should return just the direct roles
    - User may have '*multiple direct roles'* defined and there exists a hierarchy between the roles defined for the user to some other roles.
    - Function should consider all these direct roles as well as the hierarchy of these roles.
    - Here, User 2 has two direct roles 3,4   And there exists a hierarchy between **3->5->3** (There exists a circular hierarchy between 3->5->3).
    - The final list of roles is thus [3, 4, 5].

        -
          ```
          authorized_roles(2, Y).
          Y = [3,4,5];
          no
          ```
    - Similar to the above example User 3 has roles 3,4,5.

        -
          ```
          authorized_roles(3, Y).
          Y = [3,4,5];
          no
          ```

Method Used:

- We first obtain the List of the direct roles of a given User as defined by the ur(x, y) function.
    - To do this we take the set of the rules that the 'ur(x, y)' function returns.
- For each role in the list of the direct roles, we get the list of the all its descendant roles.
    - In order to find the descendants, we first defined an role_list and member_list functions which basically check the descendant roles for the given role and the member_list function checks that whether we already visited this particular edge or not. I.e, it checks whether there exists a cycle or not. The role_list function is called by descended_roles function which finally returns the all the roles of the given user. This descended function is called by authorized_roles for the given user.
- We then sort the obtained result.

- **Handling Circularity:** In order to handle circularity, we maintained information about the roles that have already been considered from the hierarchy and considered a certain hierarchy only if that particular role wasn't already processed. This was checked in the member list function which is a recursive function which checks whether the given role is already there in the list.

Question 2:

For Question 2, we are required to return all the authorized permissions for a given User.

Cases Considered:

- User may not have any role defined, thus will not have any permissions defined.
- User has roles defined, but these roles have no permissions defined.
    - In both the above cases, the function returns nothing, meaning User has no permissions defined.
    - User may have multiple roles, and these roles may have multiple permissions.
    - Function returns a list of all these permissions.
    - User permission for all users :

        -
          ```
          | ?- authorized_permissions(2,Y).
          Y = [2,5];
          no
          ```

        -
          ```
          | ?- authorized_permissions(1,Y).
          Y = [2,4,5];
          no
          ```

        -
          ```
          | ?- authorized_permissions(3,Y).
          Y = [2,5];
          no
          ```

        -
          ```
          | ?- authorized_permissions(4,Y).
          Y = [2,5];
          no
          ```

Method Used:

- First, used the previous authorized_roles function to obtain the roles for a particular user.
- For each role, obtained a set of the List of Permissions that the user has.

Question 3:

For Question 3, we are required to return the minimum number of roles that define every user.

Output:

-
  ```
  | ?- minRoles(S).
  S = 2;
  no
  ```

Method Used:

- First, we get the count of the total no.of user defined. This is done by iterating through all users.
- Then, we iterate for each of this user (say from 5 to 1), and get the List of their permissions.
- We then compute a Set of these List of Permissions to see the unique set of permissions that define the users.
- The length of this set of list_permissions is nothing but the minimum no.of roles required to describe every user.
- In order to compute the Set Of the List of permissions, we sort the List of permissions so that List of permissions like [1, 5, 3] and [3, 1, 5] are considered as the same set.

Some assumptions:

- Let us consider a case, where the role does not have any permissions but user has that role. So in that scenario, we don't consider those roles when computing minroles because when there are no permissions given to the user, so no use of that role.
- However, some people might consider these roles having empty set of permissions and still count in minimum roles to cover all users.
- So, here the result might vary from those considering the latter case.

Some learnings used in coding:

- Because we wanted a unique set of permissions list, we used **setof** function as it gives a unique list as the output. So, it made it easier.
- Cycle can be detected by member function.
- We imported some functions like between, member, append and sort.