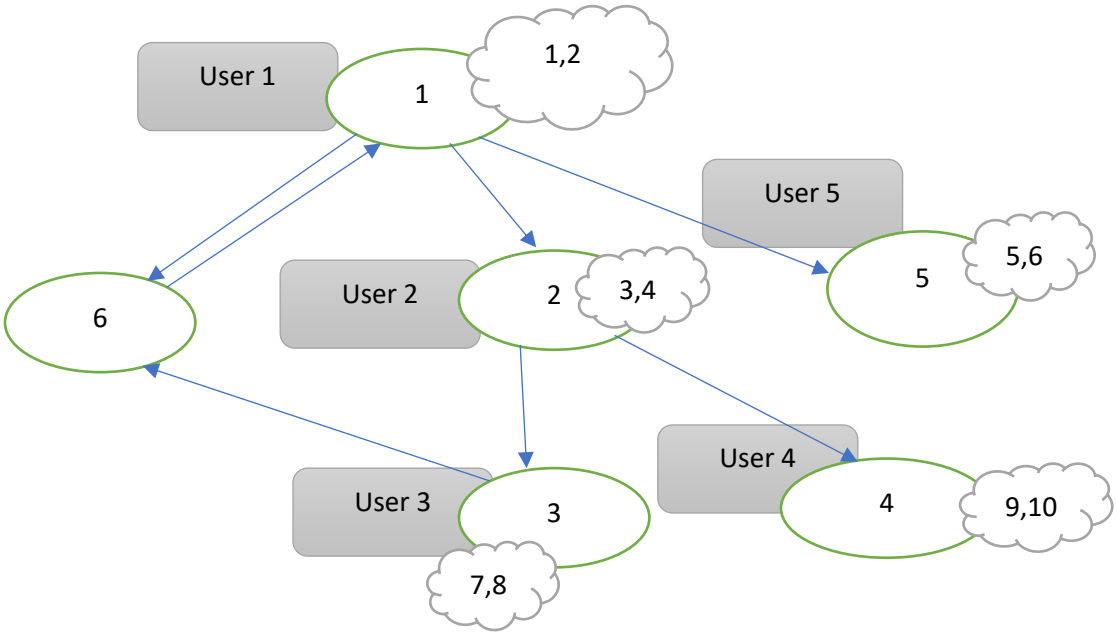
Project 03: Role Based Access Control (RBAC)
Sunny Shah- 112044068

1. Implementation Details:

- The submitted code works even if there is **a cycle made by a role hierarchy**.
- Predicate **authorized_roles** will return all unique roles (ones directly assigned using ur mapping and all their descendants) assigned to a user.
- Predicate **authorized_permissions** will return all unique permissions (ones from roles directly assigned using ur mapping and permissions from all the descendant roles) assigned to a user.
- **minRoles** will try to find minimum roles that should have been created such that the permission requirement of each user can be satisfied.

2. Knowledge Base

Consider the below knowledge base visualized in a graphical format.



In the above diagram, Rectangle denote users, Circles denotes Roles, Arrows denote hierarchy, clouds denote permissions to each circled role.

From the graph, we can see that:

User Number	*Roles*	*Permissions*
1	[1,2,3,4,5,6]	[1,2,3,4,5,6,7,8,9,10]
2	[1,2,3,4,5,6]	[1,2,3,4,5,6,7,8,9,10]
3	[1,2,3,4,5,6]	[1,2,3,4,5,6,7,8,9,10]
4	[4]	[9,10]
5	[5]	[5,6]

MinRoles = 3:

We can assign user 1,2,3 the same role as they have same permissions and user 4,5 2 different roles. So, the answer should be 3.

3. Execution of the test cases:

Question 1: authorized_roles

- Running authorized_roles for each user in the model.

```
C:\Users\sunny\Documents\AI_PROLOG>C:\"Program Files
(x86)"\XSB\config\x86-pc-windows\bin\xsb.exe
[xsb_configuration loaded]
[sysinitrc loaded]
[xsbbrat loaded]

XSB Version 3.8.0 (Three-Buck Chuck) of October 28, 2017
[x86-pc-windows; mode: optimal; engine: slg-wam; scheduling: local]
[Build date: 2017-10-31]

| ?- [test].
[Compiling .\test]
[test compiled, cpu time used: 0.1090 seconds]
[test loaded]
Yes

| ?- [project3].
[Compiling .\project3]
++Warning[XSB]: [Compiler] .\project3 : Singleton variable S in a
clause of role_heirarchy/3
++Warning[XSB]: [Compiler] .\project3 : Singleton variable S in a
clause of auth_role/3
++Warning[XSB]: [Compiler] .\project3 : Singleton variable X in a
clause of findlen/2
[project3 compiled, cpu time used: 0.0320 seconds]
[project3 loaded]
```

```
Yes

| ?- authorized_roles(1,X).
X = [1,2,3,4,5,6]
Yes
```

Analysis for the above output:

From the graph, we can see that role 1 inherits from each of the other role and ends up in a hierarchy cycle. The correct output is thus the list of all roles.

```
| ?- authorized_roles(2,X).
X = [1,2,3,4,5,6]
Yes
```

Analysis for the above output:

From the graph, we can see that role 2 inherits from each of the other role and ends up in a hierarchy cycle. The correct output is thus the list of all roles.

```
| ?- authorized_roles(3,X).
X = [1,2,3,4,5,6]
Yes
```

Analysis for the above output:

From the graph, we can see that role 3 inherits from each of the other role and ends up in a hierarchy cycle. The correct output is thus the list of all roles.

```
| ?- authorized_roles(4,X).
X = [4]
Yes
```

Analysis for the above output:

From the graph, we can see that role 4 does not inherit any other roles. The correct output is thus [4].

```
| ?- authorized_roles(5,X).
X = [5]
Yes
```

Analysis for the above output:

From the graph, we can see that role 4 does not inherit any other roles. The correct output is thus [5].

Question 2: authorized_permissions

- Running authorized_permissions for each user in the model.

```
| ?- authorized_permissions(1,X).
X = [1,2,3,4,5,6,7,8,9,10]
```

```
Yes
```

Analysis for the above output:

From the graph, we can see that role 1 inherits every other role and thus, it will also have the permissions assigned to each of those roles.

```
| ?- authorized_permissions(2,X).
X = [1,2,3,4,5,6,7,8,9,10]
Yes
```

Analysis for the above output:

From the graph, we can see that role 2 inherits every other role and thus, it will also have the permissions assigned to each of those roles.

```
| ?- authorized_permissions(3,X).
X = [1,2,3,4,5,6,7,8,9,10]
Yes
```

Analysis for the above output:

From the graph, we can see that role 3 inherits every other role and thus, it will also have the permissions assigned to each of those roles.

```
| ?- authorized_permissions(4,X).
X = [9,10]
Yes
```

Analysis for the above output:

From the graph, we can see that role 4 will only have the permissions assigned role 4 i.e. [9,10].

```
| ?- authorized_permissions(5,X).
X = [5,6]
Yes
```

Analysis for the above output:

From the graph, we can see that role 4 will only have the permissions assigned role 5 i.e. [5,6].

Question 3: minRoles

```
| ?- minRoles(X).
X = 3
yes
```

Analysis for the above output:

Below graph details the permissions required by each user.

- From the table, we see that User 1, 2, 3 share the same permissions and thus can be assigned same role.
- User 4 requires permission 9 and 10 and thus we will create a different role for him.

- User 5 requires permission 5 and 6 and thus we will create a different role for him.

Therefore, we will need to create minimum 3 roles to satisfy the permission requirements of each user.

User Number	*Permissions*
1	[1,2,3,4,5,6,7,8,9,10]
2	[1,2,3,4,5,6,7,8,9,10]
3	[1,2,3,4,5,6,7,8,9,10]
4	[9,10]
5	[5,6]