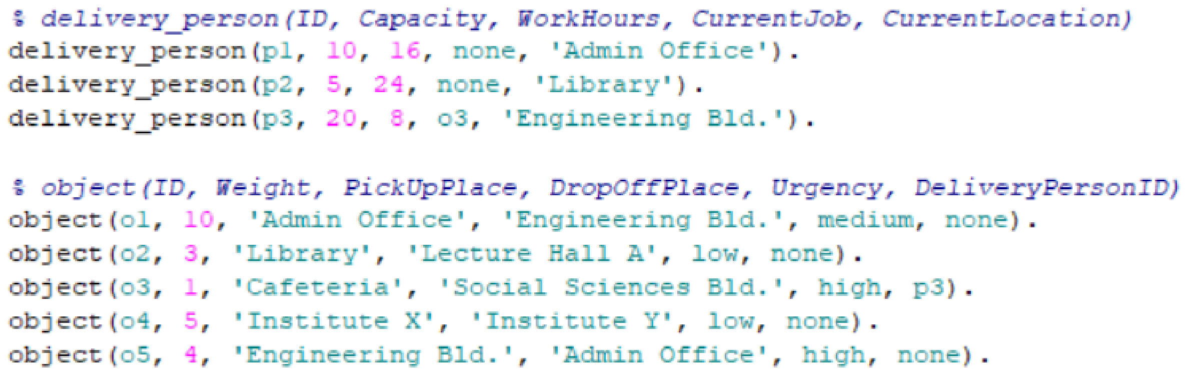
Ertugrul Kasikci 200104004097

CSE 341 HW4

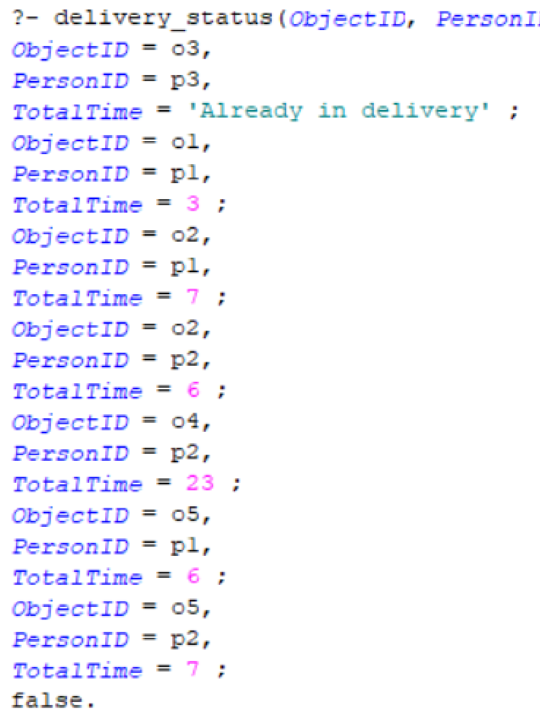
Part 1:

The program finds the minimum distance possible and the route is determined according to this. Here the current delivery persons and objects in the program.



```
% delivery_person(ID, Capacity, WorkHours, CurrentJob, CurrentLocation)
delivery_person(p1, 10, 16, none, 'Admin Office').
delivery_person(p2, 5, 24, none, 'Library').
delivery_person(p3, 20, 8, o3, 'Engineering Bld.').

% object(ID, Weight, PickUpPlace, DropOffPlace, Urgency, DeliveryPersonID)
object(o1, 10, 'Admin Office', 'Engineering Bld.', medium, none).
object(o2, 3, 'Library', 'Lecture Hall A', low, none).
object(o3, 1, 'Cafeteria', 'Social Sciences Bld.', high, p3).
object(o4, 5, 'Institute X', 'Institute Y', low, none).
object(o5, 4, 'Engineering Bld.', 'Admin Office', high, none).
```



```
?- delivery_status(ObjectID, PersonID, TotalTime).
ObjectID = o3,
PersonID = p3,
TotalTime = 'Already in delivery' ;
ObjectID = o1,
PersonID = p1,
TotalTime = 3 ;
ObjectID = o2,
PersonID = p1,
TotalTime = 7 ;
ObjectID = o2,
PersonID = p2,
TotalTime = 6 ;
ObjectID = o4,
PersonID = p2,
TotalTime = 23 ;
ObjectID = o5,
PersonID = p1,
TotalTime = 6 ;
ObjectID = o5,
PersonID = p2,
TotalTime = 7 ;
false.
```

The output above shows that the program finds the minumum distance and takes the work hours and weight into account correctly.

- p3 is already delivering o3 so it can’t deliver any other object. Also any other person can’t deliver o3.
- p1 can’t deliver o4 because this would take 24 hours and p1 can work for 16 hours.
- p2 can’t deliver o1 because o1 is 10 kg which is higher than p2’s capacity.

This query shows the program’s capabilities well. But other specific queries can be made to see it it works.

Part 2:

The decision tree for IRIS data set is implemented in python and the rules are printed using;

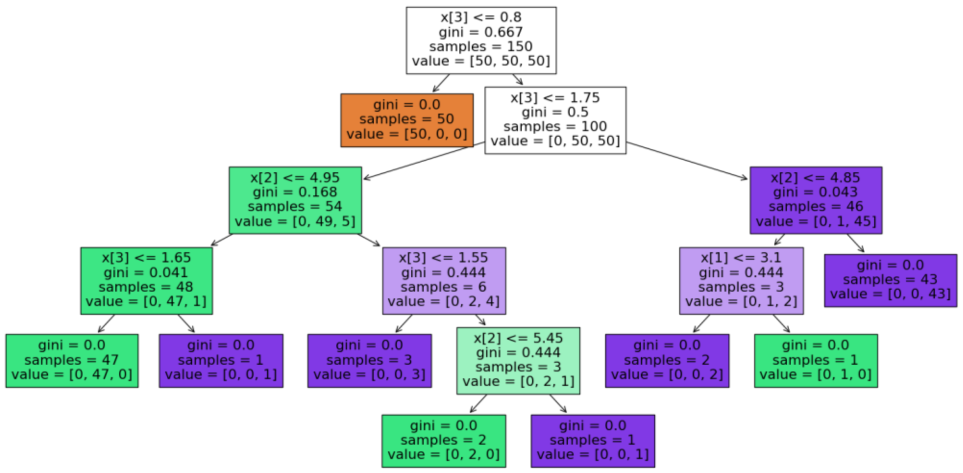
```
from sklearn.tree import export_text

decision_rules = export_text(tree_clf, feature_names=iris.feature_names)

print(decision_rules)
```

The python file also included in the project folder.

The output tree image:



The output tree rules:

```
|--- feature_3 <= 0.80
|   |--- class: Iris-setosa
|--- feature_3 >  0.80
|   |--- feature_3 <= 1.75
|   |   |--- feature_2 <= 4.95
|   |   |   |--- feature_3 <= 1.65
|   |   |   |   |--- class: Iris-versicolor
|   |   |   |--- feature_3 >  1.65
|   |   |   |   |--- class: Iris-virginica
|   |   |--- feature_2 >  4.95
|   |   |   |--- feature_3 <= 1.55
|   |   |   |   |--- class: Iris-virginica
|   |   |   |--- feature_3 >  1.55
|   |   |   |   |--- feature_2 <= 5.45
|   |   |   |   |   |--- class: Iris-versicolor
|   |   |   |   |--- feature_2 >  5.45
|   |   |   |   |   |--- class: Iris-virginica
|   |--- feature_3 >  1.75
|   |   |--- feature_2 <= 4.85
|   |   |   |--- feature_1 <= 3.10
|   |   |   |   |--- class: Iris-virginica
|   |   |   |--- feature_1 >  3.10
|   |   |   |   |--- class: Iris-versicolor
|   |   |--- feature_2 >  4.85
|   |   |   |--- class: Iris-virginica
```

Here;

Feature 0: Sepal Length

Feature 1: Sepal Width

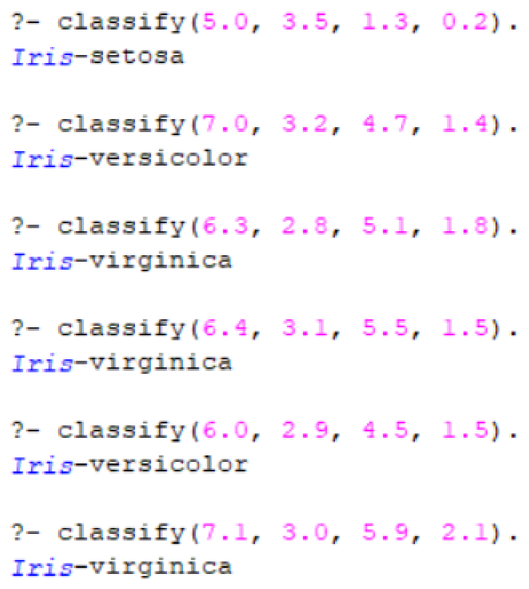
Feature 2: Petal Length

Feature 3: Petal Width

According to these rules the prolog program is implementd.

classify(SepalLength, _, PetalLength, PetalWidth) :- (Sepal Width was not used as it has no effect on the decision tree)

The output results:



```
?- classify(5.0, 3.5, 1.3, 0.2).
Iris-setosa

?- classify(7.0, 3.2, 4.7, 1.4).
Iris-versicolor

?- classify(6.3, 2.8, 5.1, 1.8).
Iris-virginica

?- classify(6.4, 3.1, 5.5, 1.5).
Iris-virginica

?- classify(6.0, 2.9, 4.5, 1.5).
Iris-versicolor

?- classify(7.1, 3.0, 5.9, 2.1).
Iris-virginica
```