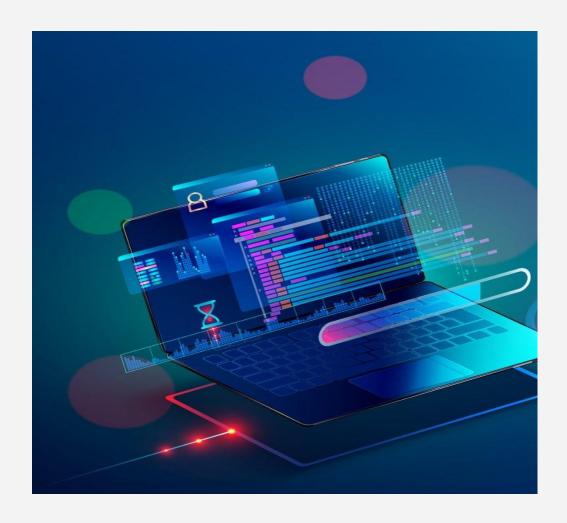
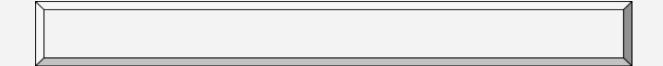
HY487 Assignment 4



Γιώργος Γεραμούτσος csd3927



Task A

The rule interconnected(X, Y):- interconnected(Y, X) would be problematic for representing a non-directed graph. Since this rule is recursive it means that it refers to itself in its definition. This means that when a programme would try to satisfy the rule it would try to find a solution for interconnected(Y, X) before it can determine the solution for interconnected(X, Y) and this would lead to an infinite loop. Using the 2 rules would be a better way to represent the bi-directional relationship because they can be satisfied individually. In some situations the recursive single rule could be used if in a controlled manner. For example if used in conjunction with another rule that would cause it to break out of the recursive calls at some point.

Note that Task B and C are implemented for the non-directed graph in figure 1b

Task B

```
🐼 SWISH
                   File ▼
                         Edit → Examples →
                                                Help▼
 🔯 🛦 Program 🗶 🛨
   1 interconnected(X, Y) :- connect(X, Y).
   2 interconnected(X, Y) :- connect(Y, X).
   4 connect(a, b).
   5 connect(b, c).
   6 connect(b, d).
   7 connect(c, f).
   8 connect(d, c).
   9 connect(f, e).
  10 connect(e, d).
  11
  12 exists_path(Inital, Goal) :-
  13
        Inital = Goal,
        . .
  15 exists_path(Inital, Goal) :-
  16
         interconnected(Inital, Goal),
  17
  18 exists_path(Inital, Goal) :-
  19
        (interconnected(Inital, Next); interconnected(Goal, Prev)),
  20
         exists_path(Next, Prev).
exists_path(a,f).
true
true
true
exists_path(f,a).
true
true
true
exists_path(e,c).
true
true
true
true
true
exists_path(b,e).
true
true
true
true
true
```

Task C

```
1 %define the facts and rules
2 interconnected(X, Y) :- connect(X, Y).
3 interconnected(X, Y) :- connect(Y, X).
5 connect(a, b).
6 connect(b, c).
7 connect(b, d).
8 connect(c, f).
9 connect(d, c).
0 connect(f, e).
1 connect(e, d).
2
4 path(Initial, Goal, Route) :-
      path_aux(Initial, Goal, [Goal], Route).
5
7 path_aux(Initial, Goal, Visited, [Initial|Visited]) :-
8
      interconnected(Initial, Goal).
9
path_aux(Initial, Goal, Visited, Route) :-
1
      interconnected(Intermediate, Goal),
2
      not(member(Intermediate, Visited)),
3
      Intermediate \== Initial,
4
5
      path aux(Initial, Intermediate, [Intermediate | Visited], Route).
6
```

```
path(f,a,Route)
Route = [f, c, b, a]
Route = [f, e, d, c, b, a]
Route = [f, e, d, b, a]
Route = [f, c, d, b, a]
path(a,f,Route)
Route = [a, b, c, f]
Route = [a, b, d, c, f]
Route = [a, b, d, e, f]
Route = [a, b, c, d, e, f]
path(b,e,Route)
Route = [b, c, f, e]
Route = [b, d, c, f, e]
Route = [b, d, e]
Route = [b, c, d, e]
path(c,e,Route)
Route = [c, f, e]
Route = [c, d, e]
Route = [c, b, d, e]
```

Task D

```
1 %define the facts and rules
2 interconnected(X, Y, C) :- connect(X, Y, C).
3 interconnected(X, Y, C) :- connect(Y, X, C).
5 connect(a, b, 3).
6 connect(b, c, 12).
7 connect(b, d, 4).
8 connect(c, f, 11).
9 connect(d, c, 7).
10 connect(f, e, 15).
11 connect(e, d, 5).
13 % cost_path rule that uses the auxiliary rule to account for visited nodes and path cost
14 cost_path(Initial, Goal, Route, Cost) :-
      cost_path_aux(Initial, Goal, [Goal], 0, Route, Cost).
17 % auxiliary cost path rule accounting for visited nodes and path cost
18 cost_path_aux(Initial, Goal, Visited, CostSoFar, [Initial|Visited], CurrentCost) :-
      interconnected(Initial, Goal, Cost),
      CurrentCost is CostSoFar + Cost.
20
21
22 cost_path_aux(Initial, Goal, Visited, CurrentCost, Route, TotalCost) :-
       interconnected(Intermediate, Goal, Cost), % Check if intermediate node exists and get its cost
      not(member(Intermediate, Visited)), % Check if intermediate not already visited
24
25
      Intermediate \== Initial,
26
     NewCost is CurrentCost + Cost, % Update the current path cost
27
     % Add Intermediate node in visited list and recursively call
28
      cost_path_aux(Initial, Intermediate, [Intermediate| Visited], NewCost, Route, TotalCost).
29
```

cost_path(a,f,Route,Cost).	cost_path(b,e,Route,Cost).
Cost = 26,	Cost = 38,
Route = [a, b, c, f]	Route = [b, c, f, e]
Cost = 25,	Cost = 37,
Route = [a, b, d, c, f]	Route = [b, d, c, f, e]
Cost = 27,	Cost = 9,
Route = [a, b, d, e, f]	Route = [b, d, e]
Cost = 42,	Cost = 24,
Route = [a, b, c, d, e, f]	Route = [b, c, d, e]
cost_path(f,a,Route,Cost).	cost_path(a,e,Route,Cost).
Cost = 26,	Cost = 41,
Route = [f, c, b, a]	Route = [a, b, c, f, e]
Cost = 42,	Cost = 40,
Route = [f, e, d, c, b, a]	Route = [a, b, d, c, f, e]
Cost = 27,	Cost = 12,
Route = [f, e, d, b, a]	Route = [a, b, d, e]
Cost = 25,	Cost = 27,
Route = [f, c, d, b, a]	Route = [a, b, c, d, e]