DATA MODELLING IN ENGINEERING

MODELING BASED ON LOGIC PROGRAMMING – PART IV -

Ana Inês Oliveira aio@fct.unl.pt

2024 - 2025

NOVA FACULDADE DE CIÊNCIAS E TECNOLOGIA

MDE-8

Contents

- **PROLOG**
  - Unification
  - Backtracking mechanism
  - Recursion mechanism
  - Facts / Rules / Queries / Structures / Combined Queries
  - Changing the memory of PROLG
  - INPUT / OUTPUT
  - Directed Graph
  - Lists in Prolog
  - Lists and Graphs
  - Bi-directional graph
  - CUT (!)
  - INPUT / OUTPUT (part2)

Lists in PROLOG

... from previous class ...

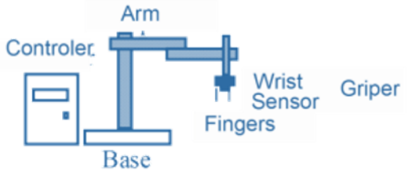
List: a data structure that can contain a variable number of elements

Some notation:

[]	empty list
[a]	list with 1 element
[a, b]	list with 2 elements
[a, [b, c], d]	list with 3 elements
[a, date(11,3,94), b]	list with 3 elements
[H | R]	list with at least 1 element H – first element (head) R - list with remaining elements (excluding head)
[X1, X2 | R]	list with at least 2 elements

Example:

```
components(robot, [base, arm, gripper, controller]).
components(gripper, [wrist, fingers, sensor]).
```




```
?-components(robot, L).
L = [base, arm, gripper, controller]

?-components(gripper, [C|R]).
C = wrist
R = [finger, sensor]
```

Example: Is the element E a **member** of a given list?

```
is_member(E, [E|_]).
is_member(E, [_|R]) :- is_member(E,R).
```

```
components(robot, [base, arm, gripper, controller]).
components(gripper, [wrist, fingers, sensor]).
```

```
?-components(robot,L), is_member(E,L).
L = [base, arm, gripper, controller],
E = base ;
L = [base, arm, gripper, controller],
E = arm ;
L = [base, arm, gripper, controller],
E = gripper ;
L = [base, arm, gripper, controller],
E = controller ;
false.
```

```
has(O,C) :- components(O,L), is_member(C,L).

?-has(gripper, fingers).
true
```

Lists in PROLOG

... from previous class ...

Example: Is the element E a **member** of a given list?

```
is_member(E, [E|_]).
is_member(E, [_|R]) :- is_member(E,R).
```

```
components(robot, [base, arm, gripper, controller]).
components(gripper, [wrist, fingers, sensor]).
```

```
?-components(robot,L), is_member(E,L).
L = [base, arm, gripper, controller],
E = base ;
L = [base, arm, gripper, controller],
E = arm ;
L = [base, arm, gripper, controller],
E = gripper ;
L = [base, arm, gripper, controller],
E = controller ;
false.
```

```
has(O,C) :- components(O,L), is_member(C,L).

?-has(gripper, fingers).
true
```

member(X,List)

length(List,N)

max_list(List,N) and min_list(List,N)

Apped(List1,List2,List1andList2)

reverse(List,ReverseList)

(…)

https://www.swi-prolog.org/pldoc/man?section=lists

SWI Prolog

library(lists): List Manipulation

HOME | DOWNLOAD | DOCUMENTATION | TUTORIALS | COMMUNITY | COMMERCIAL | WIKI

Documentation
Reference manual
The SWI-Prolog library
library(aggregate): Aggregation op
library(ansi_term): Print decorate
library(apply): Apply predicates o
library(assoc): Association lists
library(broadcast): Broadcast and
library(charsio): I/O on Lists of Cha
library(check): Consistency checki
library(clpb): CLP(B): Constraint L
library(clpfd): CLP(FD): Constraint
library(clpqr): Constraint Logic Pr
library(csv): Process CSV (Comma
library(dcg/basics): Various gener
library(dcg/high_order): High ord
library(debug): Print debug messa
library(dicts): Dict utilities
library(error): Error generating su
library(exceptions): Exception clas
library(fastrw): Fast reading and w
library(gensym): Generate unique
library(heaps): heaps/priority que
library(increval): Incremental dyn
library(intercept): Intercept and si
library(iostream): Utilities to deal
library(listing): List programs and

A.25 library(lists): List Manipulation

Compatibility
Virtually every Prolog system has `library(lists)`, but the set of provided predicates is diverse. There is a fair agreement on the semantics of most of these predicates, although error handling may vary.

This library provides commonly accepted basic predicates for list manipulation in the Prolog community. Some additional list manipulations are built-in. See e.g., memberchk/2, length/2.

The implementation of this library is copied from many places. These include: "The Craft of Prolog", the DEC-10 Prolog library (LISTRO.PL) and the YAP lists library. Some predicates are reimplemented based on their specification by Quintus and SICStus.

member(*?Elem, ?List*)

True if *Elem* is a member of *List*. The SWI-Prolog definition differs from the classical one. Our definition avoids unpacking each list element twice and provides determinism on the last element. E.g. this is deterministic:

```
member(X, [One]).
```

author
Gertjan van Noord

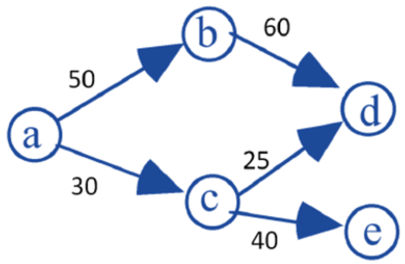
append(*?List1, ?List2, ?List1AndList2*)

List1AndList2 is the concatenation of *List1* and *List2*

Lists and Graphs

... from previous class ...

Graph revisited:



dist(a,b,50).
dist(a,c,30).
dist(b,d,60).
dist(c,d,25).
dist(c,e,40).

Obtain, in a list, the arcs of the **path** between two nodes X, Y

Solution 1:

R1 path(X,Y, [dist(X,Y,D)]) :- dist(X,Y,D).
R2 path(X,Y, [dist(X,Z,D) | R]) :- dist(X,Z,D), path(Z,Y,R).

?- path(a, d, P).

R1=> path(a, d, [dist(a, d, D)]) :- dist(a, d, D) => fails

R2=> path(a, d, [dist(a,Z,D) | R]):- dist(a,Z,D), path(Z,d,R)

Z=b, D=50

R1=> path(b, d, [dist(b, d, D')]) :- dist(b, d, D')

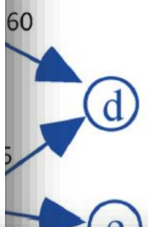
Obtain, in a list, the arcs of the path between two nodes X, Y

Solution 2:

path(X,Y, [via(X,Y)]) :- dist(X,Y,_).
path(X,Y, [via(X,Z) | R]) :- dist(X,Z,_), path(Z,Y,R).

?- path(a, d, P).

P = [via(a,b), via(b,d)] ;
P = [via(a,c), via(c,d)]



olution 3:

path(X,Y, [(X,Y)]) :- dist(X,Y,_).
path(X,Y, [(X,Z) | R]) :- dist(X,Z,_), path(Z,Y,R).

?- path(a, d, P).

P = [(a,b), (b,d)] ;
P = [(a, c), (c, d)] ;
false.

There are no more solutions

Write another version of path with the following behavior:

?-path3(a,e,P).
P = p([via(a, c, 30), via(c, e, 40)], 70)

Total distance

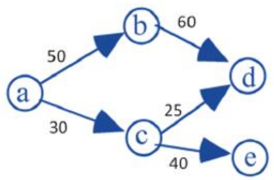
path4(X,Y, [via(X,Y,D)], D) :- dist(X,Y,D).

path4(X,Y, [via(X,Z,D1) | R], DT) :- dist(X,Z,D1), path4(Z,Y, R,D2), DT is D1+D2.

?-path4(a,d,P, D).
P= [via(a,b,50), via(b,d,60)].
D=110

path3(X,Y, p([via(X,Y,D)], D)) :- dist(X,Y,D).

path3(X,Y, p([via(X,Z,D1) | R], DT)) :- dist(X,Z,D1), path3(Z,Y, p(R,D2)), DT is D1+D2.



?- path3(a,d,P).
P = p([via(a, b, 50), via(b, d, 60)], 110) ;
P = p([via(a, c, 30), via(c, d, 25)], 55) ;
false.

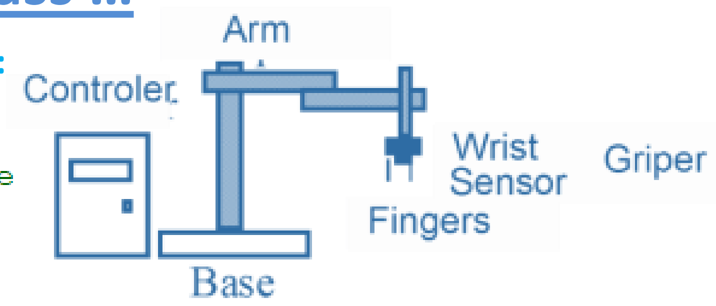
?- path3(a,e,P).
P = p([via(a, c, 30), via(c, e, 40)], 70) ;
false.

?- path3(c,e,p([via(c,e,40)],40)).
true

Lists -> Multiple Solutions: findall

... from previous class ...

Remember the Robot example:




```
% modeling robot example
part(robot,base).
part(robot,arm).
part(robot,griper).
part(robot,controler).
part(griper,wrist).
part(griper,fingers).
part(griper,sensor).
part(sensor,glass).
```

(...)

Obtaining all solutions -> **;**

```
?- part(_,P).
P = base ;
P = arm ;
P = griper ;
P = controler ;
P = wrist ;
P = fingers ;
P = sensor ;
P = glass.
```

Variable for which we want to find the value

Query, involving variable V

findall(V, query, LA)

A pre-defined rule in Prolog

List containing all possible values for V

?-findall(P,part(_,P),L).
L = [base,arm,griper,controller,wrist,fingers,sensor,glass].

?-findall(part(C,P),part(C,P),L).
L = [part(robot,base), part(robot,arm), part(robot,griper), part(robot,controller),
part(griper,wrist), part(griper,fingers), part(griper,sensor), part(sensor,glass)].

?-findall([C,P],part(C,P),L).
L = [[robot,base], [robot,arm], [robot,griper], [robot,controller],
[griper,wrist], [griper,fingers], [griper,sensor], [sensor,glass]].

findall(p(C,P),part(C,P),L).
L = [p(robot,base), p(robot,arm), p(robot,griper), p(robot,controller),
p(griper,wrist), p(griper,fingers), p(griper,sensor), p(sensor,glass)].

Lists -> Multiple Solutions: findall

```
% modeling robot example
part(robot,base).
part(robot,arm).
part(robot,griper).
part(robot,controler).
part(griper,wrist).
part(griper,fingers).
part(griper,sensor).
part(sensor,glass).
```

(…)

?-findall((C,P),part(C,P),L).
L = [(robot, base), (robot, arm), (robot, griper), (robot, controler), (griper, wrist), (griper, fingers), (griper, sensor), (sensor, glass)].

Variable for which we want to find the value

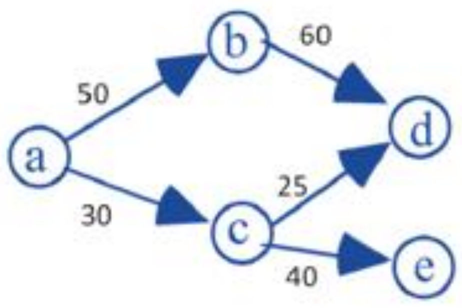
Query, involving variable V

findall(V, query, LA)

A pre-defined rule in Prolog

List containing all possible values for V

Now, let's apply it to the case of the graph:




```
dist(a,b,50).
dist(a,c,30).
dist(b,d,60).
dist(c,d,25).
dist(c,e,40).
```

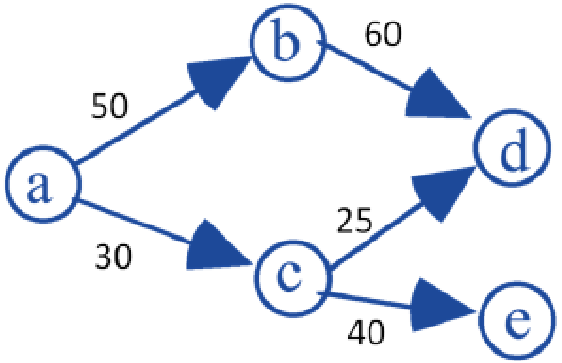
?- findall((X,Y), dist(X,Y,_), Nodes).
Nodes = [(a, b), (a, c), (b, d), (c, d), (c, e)].

?- findall(pair(X,Y), dist(X,Y,_), Nodepairs).
Nodepairs = [pair(a, b), pair(a, c), pair(b, d), pair(c, d), pair(c, e)].

?- findall([X,Y], dist(X,Y,_),L).
L = [[a, b], [a, c], [b, d], [c, d], [c, e]].

?- findall(X, path3(a,d,X), L).
L = [p([via(a, b, 50), via(b, d, 60)], 110), p([via(a, c, 30), via(c, d, 25)], 55)]

Lists -> Multiple Solutions: findall




```
dist(a,b,50).
dist(a,c,30).
dist(b,d,60).
dist(c,d,25).
dist(c,e,40).

distance(X,Y,D) :- dist(X,Y,D).
distance(X,Y,D) :- dist(X,Z,D1),
                   distance(Z,Y, D2),
                   D is D1 + D2.
```

Find the **minimal distance** between two nodes:

=> First obtain a list with all distances, i.e. considering all possible paths, and then find the minimum

```
mindist(X,Y,D) :- findall(Di, distance(X,Y,Di), LDi), min(LDi, D).

min([X],X).
min([X|R], X) :- min(R,M), X =< M.
min([X|R],M) :- min(R,M), X > M.
```

```
?- mindist(a,d,M).
M = 55
```

from @L.M. Camarinha-Matos 2023

Lists -> Multiple Solutions: findall

Write a program that returns **all paths** between 2 nodes in a graph.

```
allpaths(X,Y,AP):-findall(P,path3(X,Y,P),AP).
```

```
?- allpaths(a,d,P).
P = [p([via(a, b, 50), via(b, d, 60)], 110), p([via(a, c, 30), via(c, d, 25)], 55)].
```

Write a program that returns, in a list, the **shortest path** between 2 nodes.

```
minpath(X,Y,MP):-findall(P,path3(X,Y,P),AP), minp(AP,MP).

minp([X],X).
minp([p(X,D)|R], p(X,D)) :- minp(R,p(_,M)), D =< M.
minp([p(_,D)|R],p(Y,M)) :- minp(R,p(Y,M)), D > M.
```

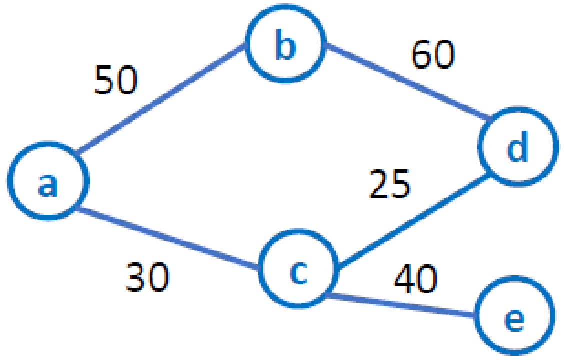
```
?- minpath(a,d,MP).
MP = p([via(a, c, 30), via(c, d, 25)], 55)
```

Suggested exercise:
E7: Do the same for the maximum distance.

from @L.M. Camarinha-Matos 2023

Bi-Directional Graph

Graph revisited – graph with **bi-directional** arcs



Solution 1: Brute force

```
dist(a,b,50).     dist(b,a,50).
dist(a,c,30).     dist(c,a,30).
dist(b,d,60).     dist(d,b,60).
dist(c,d,25).     dist(d,c,25).
dist(c,e,40).     dist(e,c,40).
```

... and we could use the same algorithms as before

Solution 2: Without repeating the arcs

```
dist(a,b,50).
dist(a,c,30).
dist(b,d,60).
dist(c,d,25).
dist(c,e,40).

biarc(X,Y,D) :- dist(X,Y,D).
biarc(X,Y,D) :- dist(Y,X,D).
```

A first attempt:

```
bi_path(X,Y,[via(X,Y,D)]) :- biarc(X,Y,D).
bi_path(X,Y,[via(X,Z,D) | R]) :- biarc(X,Z,D), bi_path(Z,Y,R).
```

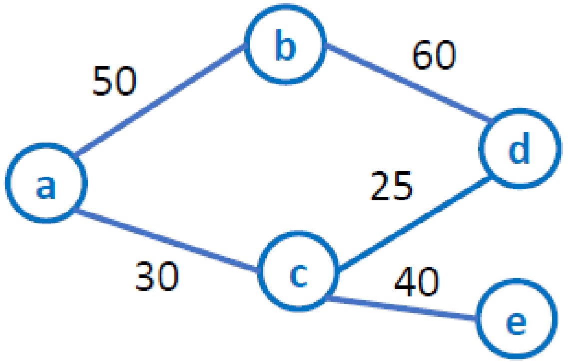
BUT: How many solutions can you get for: bi_path(a,c,L) ?
Why ?

Infinite number of solutions
-> we can pass through an arc several times !!!!

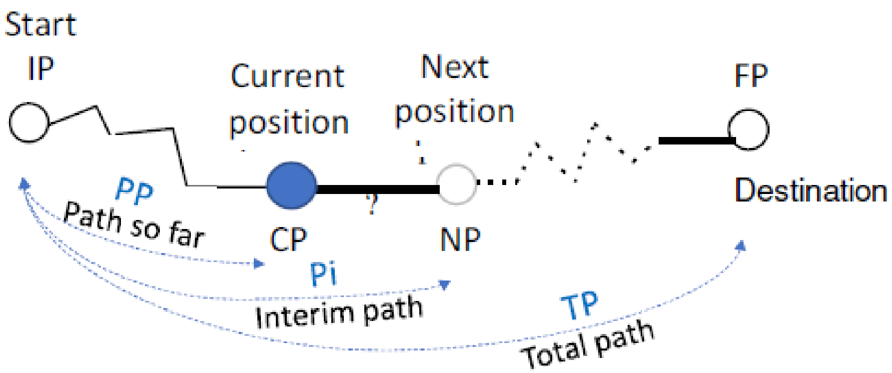
from @L.M. Camarinha-Matos 2023

Bi-Directional Graph

Solution in which we **cannot pass more than once by the same arc:**



Let's imagine a rule **step** which departs from current position CP and taking into account the path previously taken (PP), progresses to a further node (NP) towards the final destination (FD) without passing twice by the same arc.




```
R1  step(CP,FP,PP,TP) :- biarc(CP,FP,D),
                              addcond(PP, via(CP, FP, D),TP).

R2  step(CP,FP,PP,TP) :- biarc(CP,NP,D),
                              addcond(PP, via(CP, NP, D),Pi),
                              step(NP, FP, Pi, TP).
```

bipath(X,Y,TP) :- step(X,Y,[],TP).

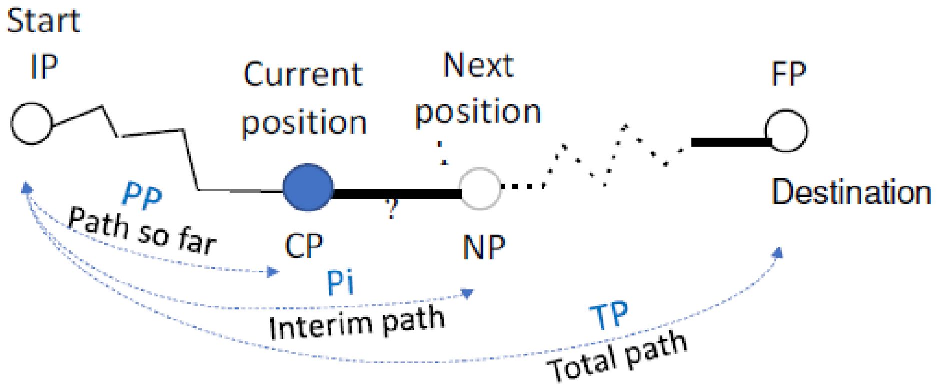
R1 is for the case that CP is directly connected to the FP

R2 is for the general case

addcond *will add a new arc to the previous path PP if and only if we never passed through that arc (in either direction)*

from @L.M. Camarinha-Matos 2023

Bi-Directional Graph




```
addcond(PP, via(P1,P2, D), Pi) :- not(member(via(P1,P2, D),PP)),
                                  not(member(via(P2,P1, D),PP)),
                                  conc(PP,[via(P1,P2, D)],Pi).
```

This rule uses the concatenation of 2 lists

```
bipath(X,Y,TP) :- step(X,Y,[],TP).

step(CP,FP,PP,TP) :- biarc(CP,FP,D),
                     addcond(PP, via(CP, FP, D),TP).

step(CP,FP,PP,TP) :- biarc(CP,NP,D),
                     addcond(PP, via(CP, NP, D),Pi),
                     step(NP, FP, Pi, TP).
```

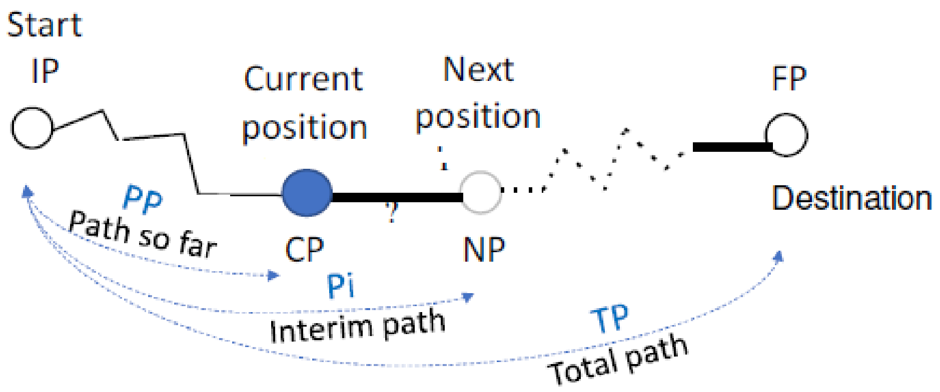
```
?- bipath(a,e,P).
P = [via(a, b, 50), via(b, d, 60), via(d, c, 25), via(c, e, 40)] ;
P = [via(a, c, 30), via(c, e, 40)] ;
false.
```

There are no more solutions

from @L.M. Camarinha-Matos 2023

Bi-Directional Graph

Which changes are needed to return both the path and its distance?




```
addcond(PP, via(P1,P2, D), Pi) :- not(member(via(P1,P2, D),PP)),
                                  not(member(via(P2,P1, D),PP)),
                                  conc(PP,[via(P1,P2, D)],Pi).
```

***step** has 2 new parameters:*

- *PD – distance run so far*
- *TD – total distance*

... when we start, PD is 0

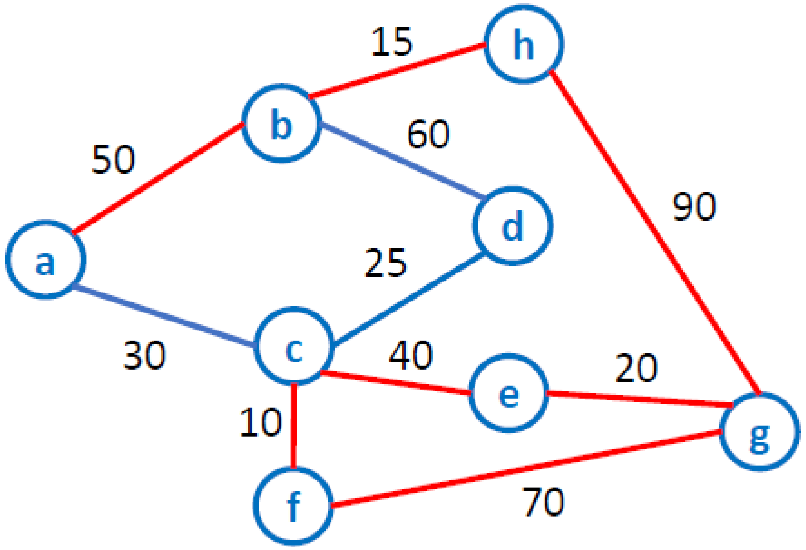
```
bipath(X,Y,TP, TD) :- step(X,Y,[],TP, 0, TD).

step(CP,FP,PP,TP, PD, TD) :- biarc(CP,FP,D),
                             addcond(PP, via(CP, FP, D),TP),
                             TD is PD + D.
step(CP,FP,PP,TP, PD, TD) :- biarc(CP,NP,D),
                             addcond(PP, via(CP, NP, D),Pi),
                             Di is PD + D,
                             step(NP, FP, Pi, TP, Di, TD).
```

```
?- bipath(a,e,P,D).
P = [via(a, b, 50), via(b, d, 60), via(d, c, 25), via(c, e, 40)],
D = 175 ;
P = [via(a, c, 30), via(c, e, 40)],
D = 70 ;
false.
```

from @L.M. Camarinha-Matos 2023

Bi-Directional Graph

But the previous algorithm still has problems – it could generate a path like a->b->h->g->f->c->e->g



=> **We should avoid passing more than once by the same node.**

```
pass_once(X,Y,TP, TD) :- stepnr(X,Y,[],TP, 0, TD).

stepnr(CP,FP,PP,TP, PD, TD) :- biarc(CP,FP,D),
                               addnorep(PP, via(CP, FP, D),TP), TD is PD + D.
stepnr(CP,FP,PP,TP, PD, TD) :- biarc(CP,NP,D),
                               addnorep(PP, via(CP, NP, D),Pi), Di is PD + D,
                               stepnr(NP, FP, Pi, TP, Di, TD).

addnorep(PP, via(P1,P2, D), Pi) :- not(passed(PP, P2)),
                                   conc(PP,[via(P1,P2, D)],Pi).

passed([via(P,_,_)|_], P).
passed([via(_,P,_)|_], P).
passed([_|R], P) :- passed(R, P).
```

Checks if we already passed by node P

```
dist(a,b,50).    dist(c,f,10).
dist(a,c,30).    dist(e,g,20).
dist(b,d,60).    dist(f,g,70).
dist(c,d,25).    dist(b,h,15).
dist(c,e,40).    dist(h,g,90).

biarc(X,Y,D) :- dist(X,Y,D).
biarc(X,Y,D) :- dist(Y,X,D).
```

```
?- pass_once(a,g,P,D).
P = [via(a, b, 50), via(b, d, 60), via(d, c, 25), via(c, e, 40), via(e, g, 20)],  D = 195 ;
P = [via(a, b, 50), via(b, d, 60), via(d, c, 25), via(c, f, 10), via(f, g, 70)],  D = 215 ;
P = [via(a, b, 50), via(b, h, 15), via(h, g, 90)],  D = 155 ;
P = [via(a, c, 30), via(c, d, 25), via(d, b, 60), via(b, h, 15), via(h, g, 90)],  D = 220 ;
P = [via(a, c, 30), via(c, e, 40), via(e, g, 20)],  D = 90 ;
P = [via(a, c, 30), via(c, f, 10), via(f, g, 70)],  D = 110 ;
false.
```

from @L.M. Camarinha-Matos 2023

Bi-Directional Graph

Obtain the shortest path and its distance

```
pass_once(X,Y,TP, TD) :- stepnr(X,Y,[],TP, 0, TD).

stepnr(CP,FP,PP,TP, PD, TD) :- biarc(CP,FP,D),
                               addnorep(PP, via(CP, FP, D),TP),
                               TD is PD + D.
stepnr(CP,FP,PP,TP, PD, TD) :- biarc(CP,NP,D),
                               addnorep(PP, via(CP, NP, D),Pi),
                               Di is PD + D,
                               stepnr(NP, FP, Pi, TP, Di, TD).

addnorep(PP, via(P1,P2, D), Pi) :- not(passed(PP, P2)),
                                   conc(PP,[via(P1,P2, D)],Pi).
passed([via(P,_,_)|_], P).
passed([via(_,P,_)|_], P).
passed([_|R], P) :- passed(R, P).
```

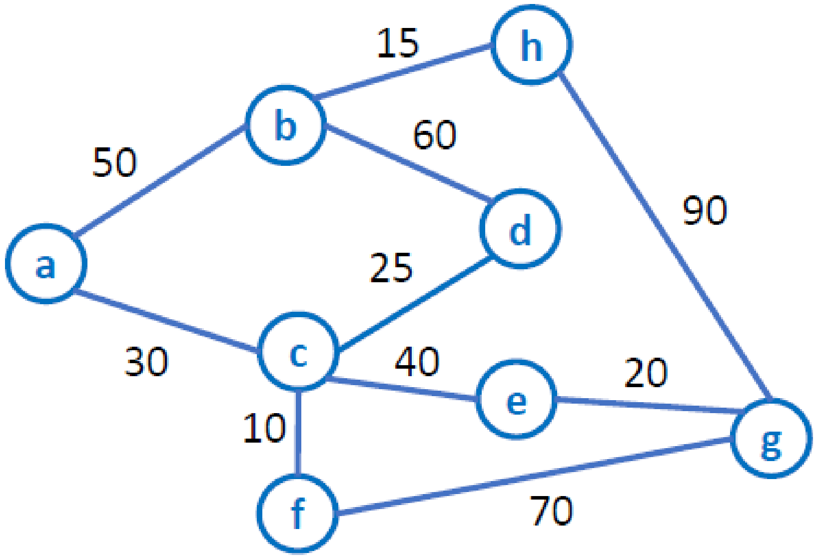
```
shortpath(X,Y,MP,MD):-findall((P,D), pass_once(X,Y,P,D),AP),
                      minp(AP,MP,MD).

minp([(P,D)],P,D).
minp([(P,D)|R], P,D) :- minp(R, _,M), D =< M.
minp([(_,D)|R],P,M) :- minp(R,P,M), D > M.
```

```
?- shortpath(a,d,P,D).
P = [via(a, c, 30), via(c, d, 25)],
D = 55

?- shortpath(d,a,P,D).
P = [via(d, c, 25), via(c, a, 30)],
D = 55

?- shortpath(g,b,P,D).
P = [via(g, h, 90), via(h, b, 15)],
D = 105
```

from @L.M. Camarinha-Matos 2023

Bi-Directional Graph

Obtain all paths between two nodes <u>that pass by another given node</u>



E.g., how to go from a to g but passing by c?

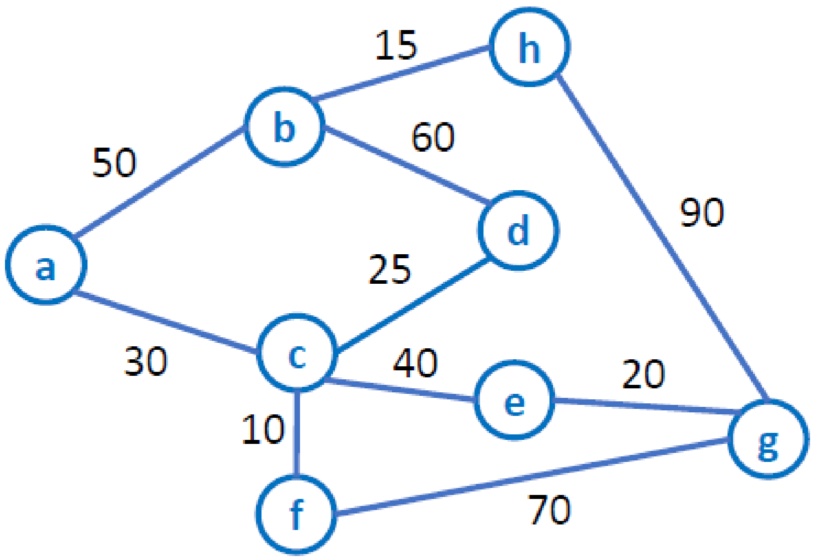
```
passnode(X,Y,I,CP):- findall(P,pass_once(X,Y,P,_),AP), filter(AP,I,CP).

filter([],_,[]).
filter([P|R],I,[P|T]):-passed(P,I),filter(R,I,T).
filter([_|R],I,T):- filter(R,I,T).
```

```
?- passnode(a,g,c,P).
P = [[via(a, b, 50), via(b, d, 60), via(d, c, 25), via(c, e, 40), via(e, g, 20)], [via(a, b, 50), via(b, d, 60), via(d, c, 25), via(c, f, 10), via(f, g, 70)], [via(a, c, 30), via(c, d, 25), via(d, b, 60), via(b, h, 15), via(h, g, 90)], [via(a, c, 30), via(c, e, 40), via(e, g, 20)], [via(a, c, 30), via(c, f, 10), via(f, g, 70)]]
```

from @L.M. Camarinha-Matos 2023

Bi-Directional Graph

What happens if we insist on finding more solutions?




```
passnode(X,Y,I,CP):- findall(P,pass_once(X,Y,P,_),AP), filter(AP,I,CP).

filter([],_,[]).
filter([P|R],I,[P|T]):-passed(P,I),filter(R,I,T).
filter([_|R],I,T):- filter(R,I,T).
```

```
?- passnode(a,g,c,P).
P = [[via(a, b, 50), via(b, d, 60), via(d, c, 25), via(c, e, 40), via(e, g, 20)],
[via(a, b, 50), via(b, d, 60), via(d, c, 25), via(c, f, 10), via(f, g, 70)], [via(a,
c, 30), via(c, d, 25), via(d, b, 60), via(b, h, 15), via(h, g, 90)], [via(a, c, 30),
via(c, e, 40), via(e, g, 20)], [via(a, c, 30), via(c, f, 10), via(f, g, 70)]] ;
P = [[via(a, b, 50), via(b, d, 60), via(d, c, 25), via(c, e, 40), via(e, g, 20)],
[via(a, b, 50), via(b, d, 60), via(d, c, 25), via(c, f, 10), via(f, g, 70)], [via(a,
c, 30), via(c, d, 25), via(d, b, 60), via(b, h, 15), via(h, g, 90)], [via(a, c, 30),
via(c, e, 40), via(e, g, 20)], [via(a, c, 30), via(c, f, 10), via(f, g, 70)]] ;
P = [[via(a, b, 50), via(b, d, 60), via(d, c, 25), via(c, e, 40), via(e, g, 20)],
[via(a, b, 50), via(b, d, 60), via(d, c, 25), via(c, f, 10), via(f, g, 70)], [via(a,
c, 30), via(c, d, 25), via(d, b, 60), via(b, h, 15), via(h, g, 90)], [via(a, c, 30),
via(c, e, 40), via(e, g, 20)]] ;
.....
```

In this case we want only one solution!

How to avoid this behavior?

from @L.M. Camarinha-Matos 2023

Cut (!)

Control of backtracking => *cut* (*!*)

Cut is a pre-defined predicate in Prolog that evaluates **true** when progressing from left to right … but does not allow backtracking

```
r :- a , b, !, c.
```

We can have normal backtracking here … if **b** fails, we can go back, find another solution for **a** and try again to prove **b**

If **c** fails, we cannot backtrack (the cut blocks it)

from @L.M. Camarinha-Matos 2023

Cut (!)

Example: Calculation of factorial

```
fact(0,1).
fact(N,F) :- N1 is N-1,
             fact(N1,F1),
             F is F1 * N.
```

```
?-fact(3,F).
F=6
```

But if we insist …

```
?- fact(3,F).
F = 6 ;
ERROR: Stack limit (1.0Gb) exceeded
ERROR:   Stack sizes: local: 1.0Gb, global: 20Kb, trail: 0Kb
ERROR:   Stack depth: 11,180,952, last-call: 0%, Choice points: 12
ERROR:   Possible non-terminating recursion:
ERROR:     [11,180,952] user:fact(-11180932, _5304)
ERROR:     [11,180,951] user:fact(-11180931, _5324)
```

A solution with cut:

```
fact(0,1) :- !.
fact(N,F) :- N1 is N-1,
             fact(N1,F1),
             F is F1 * N.
```

```
?- fact(3,F).
F = 6.

?- fact(0,F).
F = 1.
```

In this case we could also have a solution without cut, imposing a condition on N in the 2nd rule:

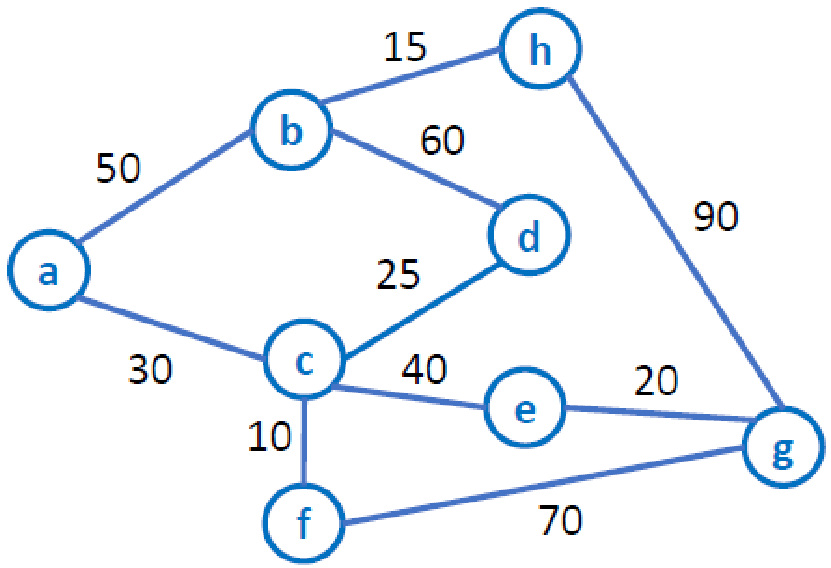
```
fact(0,1).
fact(N,F) :- N > 0, N1 is N-1, fact(N1,F1), F is F1 * N.
```

from @L.M. Camarinha-Matos 2023

FACULDADE DE CIÊNCIAS E TECNOLOGIA

Bi-Directional Graph

Going back to the program to obtain all paths between two nodes that pass by another given node




```
passnode(X,Y,I,CP):- findall(P,pass_once(X,Y,P,_),AP), filter(AP,I,CP), !.

filter([],_,[]).
filter([P|R],I,[P|T]):-passed(P,I),filter(R,I,T).
filter([_|R],I,T):- filter(R,I,T).
```

```
?- passnode(a,g,c,P).
P = [[via(a, b, 50), via(b, d, 60), via(d, c, 25), via(c, e, 40), via(e,
g, 20)], [via(a, b, 50), via(b, d, 60), via(d, c, 25), via(c, f, 10),
via(f, g, 70)], [via(a, c, 30), via(c, d, 25), via(d, b, 60), via(b, h,
15), via(h, g, 90)], [via(a, c, 30), via(c, e, 40), via(e, g, 20)],
[via(a, c, 30), via(c, f, 10), via(f, g, 70)]].
```

Only one answer, as we wanted

i.e., the cut avoids backtracking

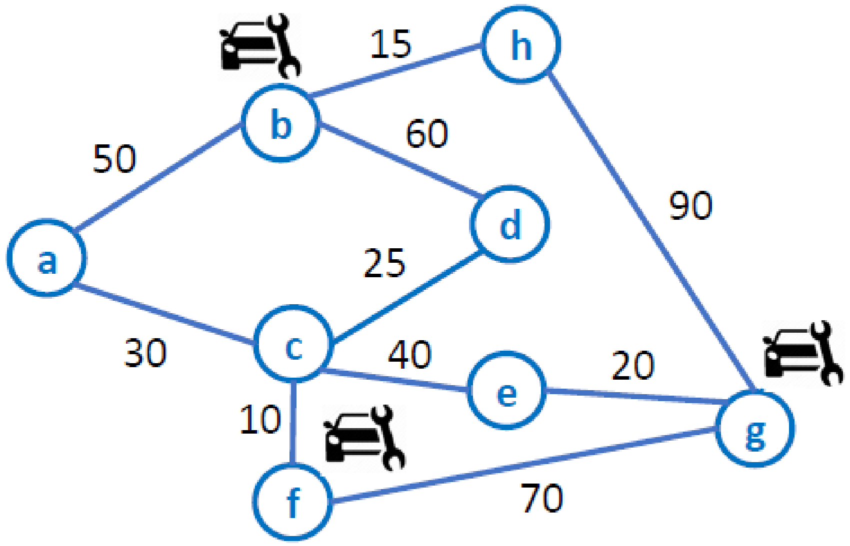
from @L.M. Camarinha-Matos 2023

Bi-Directional Graph

Suggested exercises:

E8. Obtain a route between 2 cities such that there is at least one car repair shop in that route.

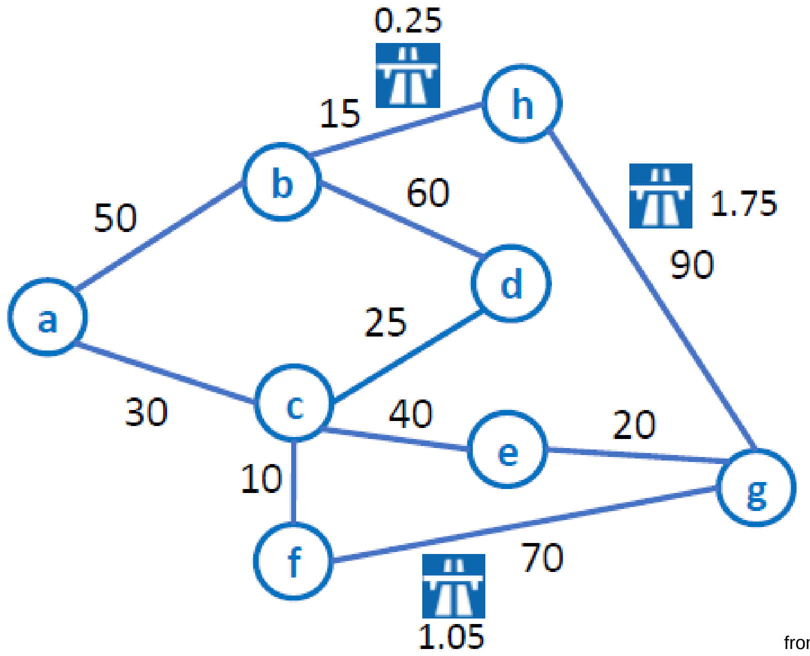
Hint: use facts in the form 'repair(c).' to indicate that city c has a repair shop.



E9. Imagine that some roads have toll; others are free.

a) Write a program to find a route between 2 nodes X and Y toll free (if there is one).

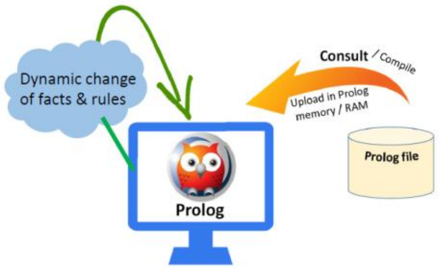
b) Write a program to calculate the cost of each route between X and Y.



from @L.M. Camarinha-Matos 2023

Input / Output

Proposed exercise:
E1: Write a program to read the elements of a graph.
E.g. read connections between cities in the form of “link(City1, City2, Distance).”



Here we use some pre-defined rules of SWI-Prolog:
- *read – reads a string ended by “.”*
- *write – writes a string*
- *nl – new line*

Something useful:

```
?- listing(dist).
dist(a, b, 50).
dist(a, c, 30).
dist(b, d, 60).
dist(c, d, 25).
dist(c, e, 40).
dist(c, f, 10).
dist(e, g, 20).
dist(f, g, 70).
dist(b, h, 15).
dist(h, g, 90).

true.
```

```
?- listing(stepnr).
stepnr(CP, FP, PP, TP, PD, TD) :-
   biarc(CP, FP, D),
   addnorep(PP, via(CP, FP, D), TP),
   TD is PD+D.
stepnr(CP, FP, PP, TP, PD, TD) :-
   biarc(CP, NP, D),
   addnorep(PP, via(CP, NP, D), Pi),
   Di is PD+D,
   stepnr(NP, FP, Pi, TP, Di, TD).

true.
```

assert(S)
asserta(S) — *To add facts/rules*
assertz(S)

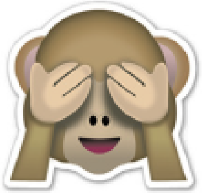
retract(S) — *To delete facts/rules*

:-dynamic fact/args

Input / Output

All Prolog implementations include a **library** of pre-defined predicates to do **input/output**.

In a previous example we already used two of these predicates:
read(S)
write(S)



However, there is no stable, standard I/O library and the set of predicates may change from implementation to implementation (as it happens with other languages)

The predicate read(S) is even not very practical, as it forces input to end with "."

But we can implement our own rules using basic I/O predicates (that operate at the level of character and are more or less standard across the various implementations of Prolog

from @L.M. Camarinha-Matos 2023

Input / Output

Some auxiliary predicates:

atom_codes(T,L) — *Converts a term T into a list L of ASCII codes, or vice-versa*

atom_chars(T, L) — *Decomposes a term T into a list L of characters, or vice-versa*

```
?- atom_codes(abc, L).
L = [97, 98, 99].

?- atom_codes('ABC', L).
L = [65, 66, 67].

?- atom_codes(S, [65, 66, 67]).
S = 'ABC' .
```

```
?- atom_chars(abc, L).
L = [a, b, c].

?- atom_chars(A, [a,b,c]).
A = abc

?- atom_chars('ABC',L).
L = ['A', 'B', 'C'].
```

from @L.M. Camarinha-Matos 2023

Input / Output

Other auxiliary predicates:

char_code(Char, Code).

Converts a character Char into its ASCII Code; or vice-versa

```
?- char_code(a, Code).
Code = 97.

?- char_code(C, 97).
C = a.

?- char_code(a,97).
true.
```

number_codes(N, LC).

Converts a number N into a list LC of ASCII codes, or vice-versa

```
?- number_codes(1,L).
L = [49]

?- number_codes(123,L).
L = [49, 50, 51]

?- number_codes(N,[50]).
N = 2.
```

For additional pre-defined predicates see:
https://www.swi-prolog.org/pldoc/man?section=manipatom from @L.M. Camarinha-Matos 2023

Input / Output

get_code(C)

Reads the current input stream and unifies C with the character code of the next character. C is unified with -1 on end of file.

```
?- get_code(C).
|  a
C = 97
```

Pressed key

get_char(C)

Reads the current input stream and unifies C with the next character as a one-character atom.

```
?- get_char(C).
|   a
C = a.
```

from @L.M. Camarinha-Matos 2023

Input / Output

put_code(C)

Write a character to the output stream corresponding to the code C

```
?- put_code(97).
a
true.
```

put_char(C)

Write a character C to the output stream

```
?- put_char(a).
a
true.
```

from @L.M. Camarinha-Matos 2023

Input / Output

get_single_char(C)

Gets a single character from input stream.
*Unlike get_code, this predicate does **not wait for a return**.*
*The character is **not echoed** to the user's terminal.*

```
?- get_single_char(C).
|   C = 97.
```

Pressed key

Can be useful to read passwords (for instance), in which we do not want to echo the characters

from @L.M. Camarinha-Matos 2023

Input / Output

Example: Read a text ended by "return" or “enter”

Solution 1:

```
read1(S) :-readlist1(L), atom_codes(S,L),!.
readlist1(L) :- get_code(C), process1(C,L).
process1(10,[]):-nl.        Enter
process1(13,[]):-nl.        Return
process1(C,[C|R]) :- readlist1(R).
```

```
?- read1(S).
|    Example

S = ‘Example'.
```

Solution 2:

```
read2(S) :-readlist2(L), atom_chars(S,L),!.
readlist2(L) :- get_char(C), process2(C,L).
process2('\n',[]):-nl.      Enter
process2('\r',[]):-nl.      Return
process2(C,[C|R]) :- readlist2(R).
```

```
?- read2(S).
|    Example

S = ‘Example'.
```

Notice the use of cut (!) to disable backtracking in case this rule is used inside another one

from @L.M. Camarinha-Matos 2023

Input / Output

Example: Write a text

Solution 1:

```
write_text(S) :- atom_chars(S,L), writelist(L), !.
writelist([]).
writelist([C|R]) :- put_char(C), writelist(R).
```

```
?- write_text('Example of text').
Example of text
true.
```

Solution 2:

```
write_text1(S) :- atom_codes(S,L), writelist1(L), !.
writelist1([]).
writelist1([C|R]) :- put_code(C), writelist1(R).
```

```
?- write_text1('Example of text').
Example of text
true.
```

from @L.M. Camarinha-Matos 2023

Input / Output

Example:
Read a password ended by “return” or “enter” .
For each pressed key, instead of echoing the corresponding symbol, display “*”.

```
readpass(P):-rpass(LP),atom_codes(P,LP),!.

rpass(LP):- get_single_char(A), processp(A,LP).
processp(10,[]):-nl.
processp(13,[]):-nl.
processp(A,[A|C]):- put_char(*), rpass(C).
```

```
?- readpass(P).
|: *******
P = example.
```

from @L.M. Camarinha-Matos 2023

Input / Output

Example:
Rule to confirma a question accepting various ways of giving a positive answer.

```
confirm(Q) :- write_text(Q), read1(R), afirmative(R).

afirmative(yes).
afirmative(y).
afirmative('Yes').
afirmative('Y').
afirmative(sure).
afirmative(yap).
afirmative('of course').
afirmative('no doubt').
```

Collection of all answers that are considered positive

```
?- confirm('Do you like Prolog?').
Do you like Prolog?yes

true.

?- confirm('Do you like Prolog?').
Do you like Prolog?Y

true.

?- confirm('Do you like Prolog?').
Do you like Prolog?of course

true.
```

from @L.M. Camarinha-Matos 2023

Input / Output

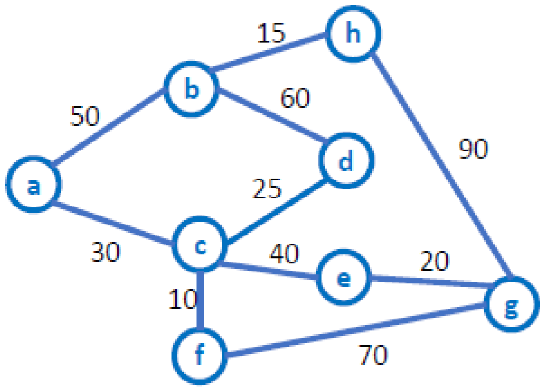
Proposed exercise:
E2: Write a Prolog predicate that receives a list as input and displays it as exemplified:

```
?-printlist([a, b, [c, d, [e, f], g], h]).

a
b
--- c
--- d
------ e
------ f
--- g
h
```

from @L.M. Camarinha-Matos 2023

Input / Output

Let's revisit a previous example:




```
dist(a,b,50).      dist(c,f,10).
dist(a,c,30).      dist(e,g,20).
dist(b,d,60).      dist(f,g,70).
dist(c,d,25).      dist(b,h,15).
dist(c,e,40).      dist(h,g,90).

biarc(X,Y,D) :- dist(X,Y,D).
biarc(X,Y,D) :- dist(Y,X,D).

conc([], L, L).
conc([C|R], L, [C|T]) :- conc(R, L, T).

pass_once(X,Y,TP, TD) :- stepnr(X,Y,[],TP, 0, TD).
```

```
stepnr(CP,FP,PP,TP, PD, TD) :- biarc(CP,FP,D), addnorep(PP, via(CP, FP, D),TP), TD is PD + D.
stepnr(CP,FP,PP,TP, PD, TD) :- biarc(CP,NP,D), addnorep(PP, via(CP, NP, D),Pi),  Di is PD + D,
                                    stepnr(NP, FP, Pi, TP, Di, TD).

addnorep(PP, via(P1,P2, D), Pi) :- not(passed(PP, P2)), conc(PP,[via(P1,P2, D)],Pi).
passed([via(P,_,_)|_], P).
passed([via(_,P,_)|_], P).
passed([_|R], P) :- passed(R, P).

passnode(X,Y,I,CP):- findall(P,pass_once(X,Y,P,_),AP), filter(AP,I,CP), !.

filter([],_,[]).
filter([P|R],I,[P|T]):-passed(P,I),filter(R,I,T).
filter([_|R],I,T):- filter(R,I,T).

?- passnode(a,g,c,P).
P = [[via(a, b, 50), via(b, d, 60), via(d, c, 25), via(c, e, 40), via(e, g, 20)], [via(a, b, 50), via(b, d,
60), via(d, c, 25), via(c, f, 10), via(f, g, 70)], [via(a, c, 30), via(c, d, 25), via(d, b, 60), via(b, h, 15),
via(h, g, 90)], [via(a, c, 30), via(c, e, 40), via(e, g, 20)], [via(a, c, 30), via(c, f, 10), via(f, g, 70)]].
```

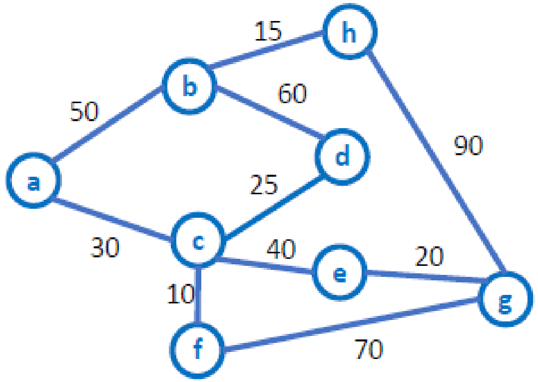
Write a program that displays the result in a more readable fashion:

```
?- nice_passnode(a,g,c).
Route 1: via(a, b, 50)-> via(b, d, 60) -> via(d, c, 25) -> via(c, e, 40) -> via(e, g, 20)
Route 2: via(a, b, 50) -> via(b, d, 60) -> via(d, c, 25) -> via(c, f, 10) -> via(f, g, 70)
Route 3: via(a, c, 30) -> via(c, d, 25) -> via(d, b, 60) -> via(b, h, 15) -> via(h, g, 90)
Route 4: via(a, c, 30) -> via(c, e, 40) -> via(e, g, 20)
Route 5: via(a, c, 30) -> via(c, f, 10) -> via(f, g, 70)
```

from @L.M. Camarinha-Matos 2023

Input / Output

Continuation ....



?- passnode(a,g,c,P).
P = [[via(a, b, 50), via(b, d, 60), via(d, c, 25), via(c, e, 40), via(e, g, 20)], [via(a, b, 50), via(b, d, 60), via(d, c, 25), via(c, f, 10), via(f, g, 70)], [via(a, c, 30), via(c, d, 25), via(d, b, 60), via(b, h, 15), via(h, g, 90)], [via(a, c, 30), via(c, e, 40), via(e, g, 20)], [via(a, c, 30), via(c, f, 10), via(f, g, 70)]].

```
nice_passnode(X,Y,I) :- passnode(X, Y, I, P), pretty_display(P).

pretty_display(P):- pdisplay(P,0).

pdisplay([X],N):- N1 is N+1, displayroute(N1,X).
pdisplay([X|R],N):-N1 is N+1, displayroute(N1,X),pdisplay(R,N1).

displayroute(N,X):- write('Route '), write(N), write(': '), droute(X), nl.

droute([X]):-write(X).
droute([X|R]):- write(X), write('-> '), droute(R).
```

```
?- nice_passnode(a,g,c).
Route 1: via(a,b,50)-> via(b,d,60)-> via(d,c,25)-> via(c,e,40)-> via(e,g,20)
Route 2: via(a,b,50)-> via(b,d,60)-> via(d,c,25)-> via(c,f,10)-> via(f,g,70)
Route 3: via(a,c,30)-> via(c,d,25)-> via(d,b,60)-> via(b,h,15)-> via(h,g,90)
Route 4: via(a,c,30)-> via(c,e,40)-> via(e,g,20)
Route 5: via(a,c,30)-> via(c,f,10)-> via(f,g,70)
true .

?- nice_passnode(a,g,f).
Route 1: via(a,b,50)-> via(b,d,60)-> via(d,c,25)-> via(c,f,10)-> via(f,g,70)
Route 2: via(a,c,30)-> via(c,f,10)-> via(f,g,70)
true
```

from @L.M. Camarinha-Matos 2023

Input / Output

Integrate some of the previous examples into a single program with a menu:

1. Enter graph
2. Show graph
3. Find shortest path
4. Pass by a node
5. Delete graph
6. Exit

```
gmenu:- nl, nl, write('GRAPH MANAGEMENT SYSTEM'), nl,
        menu(Op), execute(Op).

menu(Op):- write('1. Enter graph'),nl, write('2. Show graph'), nl,
           write('3. Find shortest path'), nl, write('4. Pass by a node'), nl,
           write(´5. Delete graph'), nl, write('6. Exit'), nl, readoption(Op).

readoption(O):-get_single_char(C),put_code(C), number_codes(O,[C]), valid(O), nl.
readoption(O):- nl, write('*** Invalid option. Try again: '), readoption(O).
valid(O):- O >=1, O=<6.

execute(6). /* exit condition*/
execute(Op):- exec(Op),nl, menu(NOp),execute(NOp).

exec(1) :- readgraph.
exec(2) :- showgraph.
exec(3) :- findspath.
exec(4) :- passnode.
exec(5) :- deletegraph.
```

from @L.M. Camarinha-Matos 2023

Input / Output

Continuation ....

```
readgraph:- nl, write('Enter arcs in the form dist(X,Y,D),
finishing with "end":'), nl, readarcs.

:- dynamic dist/3.

readarcs :- read(S), mem(S).
mem(end).
mem(dist(X,Y,D)):-assertz(dist(X,Y,D)), nl, readarcs.
mem(_):-write('Invalid data. Repeat: '), nl, readarcs.
```

```
deletegraph:- delarcs, nl, write('Graph deleted'), nl.

delarcs:- retract(dist(_,_,_)), fail.
delarcs.
```

```
showgraph:- nl, write('Graph structure: '), nl,
            listing(dist), nl.
```

```
findspath:- nl, write('Enter nodes: '), readnodes(X,Y),
shortpath(X,Y,MP,MD), displayspath(MP,MD).

readnodes(X,Y):-nl, write('begin: ‘), read1(X),
                    write('end: '),read1(Y).

shortpath(X,Y,MP,MD):-findall((P,D), pass_once(X,Y,P,D),AP),
                       minp(AP,MP,MD).
minp([(P,D)],P,D).
minp([(P,D)|R], P,D) :- minp(R, _,M), D =< M.
minp([(_,D)|R],P,M) :- minp(R,P,M), D > M.

displayspath(MP,MD):- write('Path: '), write(MP), nl,
write('Distance: '), write(MD), nl.
```

```
passnode:- nl, write('Enter nodes: '), read3nodes(X,Y,I),
nice_passnode(X,Y,I).

read3nodes(X,Y,I):- readnodes(X,Y), nl,
                    write('Intermediate node: '), read1(I).
```

from @L.M. Camarinha-Matos 2023

Input / Output

Continuation ....

```
?- gmenu.

GRAPH MANAGEMENT SYSTEM
1. Enter graph
2. Show graph
3. Find shortest path
4. Pass by a node
5. Delete graph
6. Exit
|: 2

Graph structure:
:- dynamic dist/3.

dist(a, b, 50).
dist(a, c, 30).
dist(b, d, 60).
dist(c, d, 25).
dist(c, e, 40).
dist(c, f, 10).
dist(e, g, 20).
dist(f, g, 70).
dist(b, h, 15).
dist(h, g, 90).
```

```
1. Enter graph
2. Show graph
3. Find shortest path
4. Pass by a node
5. Delete graph
6. Exit
|: 3

Enter nodes:
begin: a

end: g

Path: [via(a,c,30),via(c,e,40),via(e,g,20)]
Distance: 90
```

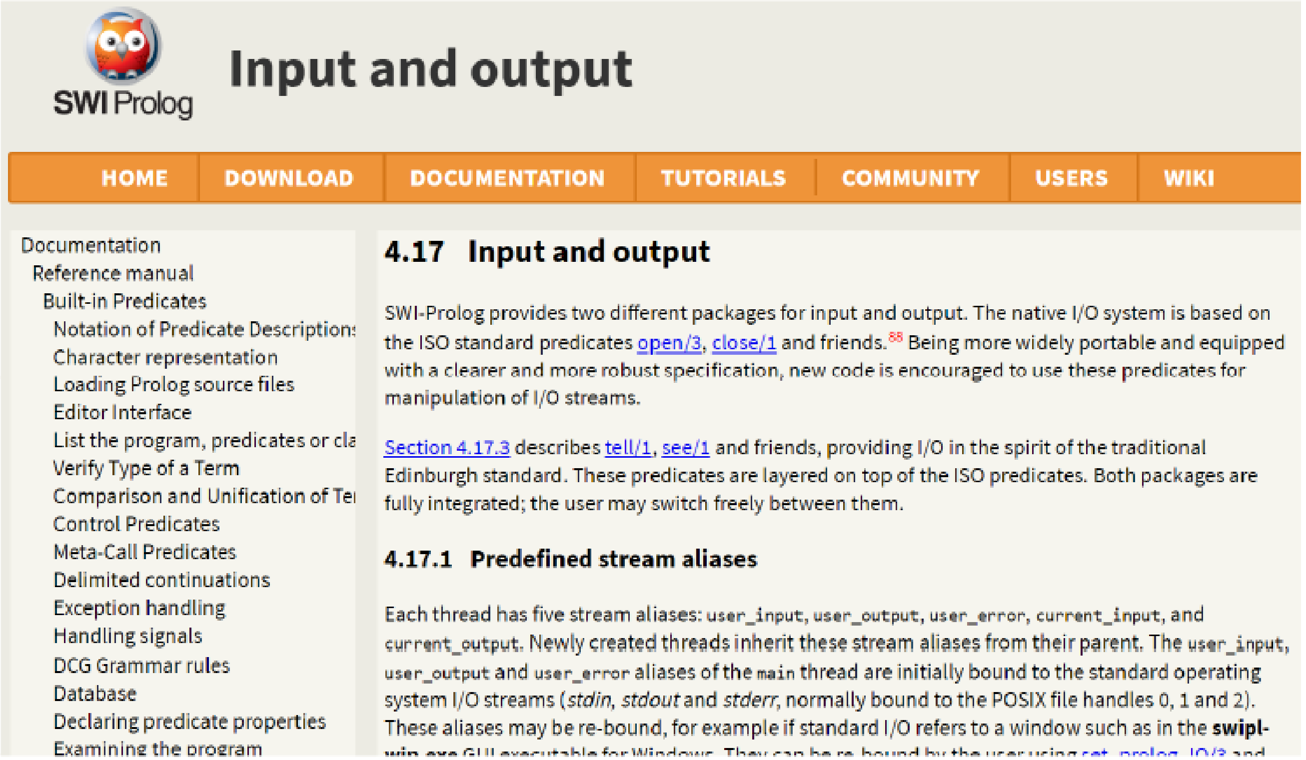
```
1. Enter graph
2. Show graph
3. Find shortest path
4. Pass by a node
5. Delete graph
6. Exit
|: 4

Enter nodes:
begin: a

end: g

Intermediate node: c

Route 1: via(a,b,50)-> via(b,d,60)-> via(d,c,25)-> via(c,e,40)-> via(e,g,20)

Route 2: via(a,b,50)-> via(b,d,60)-> via(d,c,25)-> via(c,f,10)-> via(f,g,70)

Route 3: via(a,c,30)-> via(c,d,25)-> via(d,b,60)-> via(b,h,15)-> via(h,g,90)

Route 4: via(a,c,30)-> via(c,e,40)-> via(e,g,20)

Route 5: via(a,c,30)-> via(c,f,10)-> via(f,g,70)
```

from @L.M. Camarinha-Matos 2023

Input / Output

SWI-Prolog includes an extensive **library** of I/O predicates …. including I/O from/to files

https://www.swi-prolog.org/pldoc/man?section=IO



from @L.M. Camarinha-Matos 2023

Additional Resources

Packages

HOME DOWNLOAD DOCU

Reference manual
Packages
- Paxos -- a SWI-Prolog replicating ke
- SWI-Prolog SSL Interface
- SWI-Prolog ODBC Interface
- SWI-Prolog Regular Expression libr
- Pengines: Web Logic Programming
- SWI-Prolog C-library
- Transparent Inter-Process Commur
- SWI-Prolog SGML/XML parser
- Constraint Query Language A high l
- SWI-Prolog Natural Language Proce
- SWI-Prolog Source Documentation
- JPL: A bidirectional Prolog/Java int

Package
Package
the build

See also
pack_ins

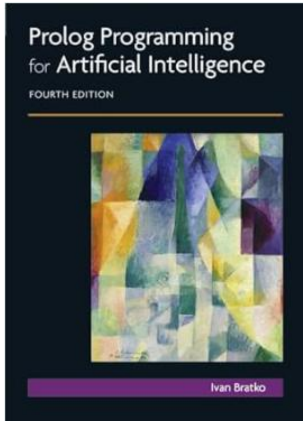
Packs (add-ons) for SWI-Prolog

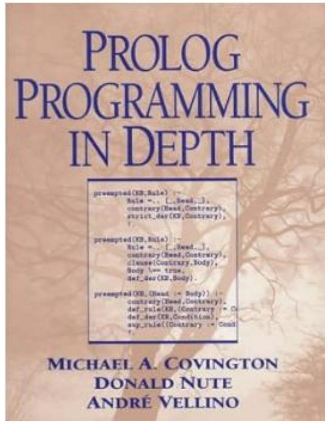
Pack tot: 308	Version (#older)	Downloads tot: 22,265 (#latest)	Rating (#votes/ #comments)	Title
achelois	0.5.0(7)	3,991(69)		Collection of tools to make writing scripts in Prolog easier.
aleph	5(19)	232(1)		Aleph Inductive Logic Programming system
amazon_api	0.0.3(2)	57(50)		Interface to Amazon APIs
ansi_termx	0.0.1	7		ANSI terminal operations
ape	6.7.0(1)	20(2)		Parser for Attempto Controlled English (ACE)
app	0.1	12		Prolog Application Server
arouter	1.1.1(4)	166(120)		Alternative HTTP path router
assertions	0.0.1(26)	38(1)		Ciao Assertions Reader for SWI-Prolog
atom_feed	0.2.0(4)	53(39)		Parse Atom and RSS feeds
auc	1.0(10)	86(2)		Library for computing Areas Under the Receiving Operating
b_real	0.4(3)	29(17)		Interface predicates to commonly used R functions.
bddem	4.3.1(16)	67(2)		A library for manipulating Binary Decision Diagrams
bencode	0.0.1	37		Bencoding from BitTorrent protocol
bibtex	0.1.6(1)	8(7)		Parser and predicates for BibTeX files
bims	2.3(5)	52(16)		Bayesian inference of model structure.
bio_analytics	0.3(2)	11(6)		Computational biology data analytics.
bio_db	3.1(19)	62(3)		Access, use and manage big, biological datasets.
bio_db_repo	20.3.8(14)	39(2)		Data package for bio_db.
biomake	0.1.5(8)	25(6)		Prolog makefile-like system
blog_core	1.5.2(22)	89(3)		Blog/CMS framework
body_reordering	1.4.111(4)	14(2)		Clause expansion Utils for deciding which order to run Goal

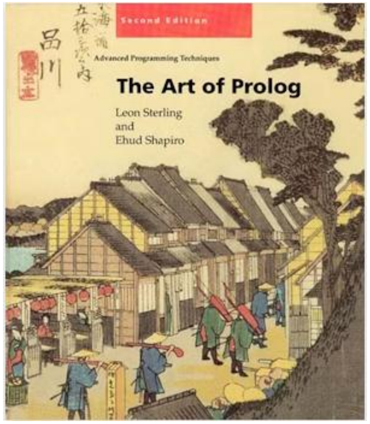
https://www.swi-prolog.org/pldoc/doc_for?object=packages

https://www.swi-prolog.org/pack/list

Further reading







(…)

https://www.amazon.com/Programming-Artificial-Intelligence-International-Computer/dp/0321417461

https://www.amazon.com/Prolog-Programming-Depth-Michael-Covington/dp/013138645X/ref=pd_sim_14_4?ie=UTF8&dpID=514M0RXA1WL&dpSrc=sims&preST=_AC_UL160_SR122%2C160_&refRID=1TM7A3CEFC2BD4JA77WR

https://mitpress.mit.edu/9780262691635/the-art-of-prolog/

SWI Prolog

https://www.swi-prolog.org/pldoc/doc_for?object=manual

WIKIBOOKS
Open books for an open world

https://en.wikibooks.org/wiki/Prolog

https://drsmithbiology.weebly.com/further-reading.html

Good Work!

Ana Inês Oliveira
NOVA School of Sciences and Technology | FCT NOVA

aio@fct.unl.pt