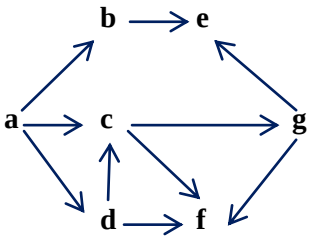
MSc Computing Science
Imperial College London

Prolog Unassessed Laboratory
November 2016

1. **The given file for this question is graph.pl. Extend it by adding your programs to it.**

The given file represents the graph below, as a set of *arc/2* facts.



Add definitions for the following relations to the file. You can define auxiliary relations if you need them. You can use *findall*, *setof*, \= and Prolog conditional and negation operators, if needed, together with any arithmetic primitives you may need (e.g. <, >), and list predicates, *member*, *length*, *append*, but no other built-in predicates. Do not use the cut, *!*.
Your programs should work correctly whatever the graph is, i.e. whatever instances are provided for the predicate *arc*. The graph above and the data in the file graph.pl is one particular set of instances.

i) **deadEnd(N)**
to mean node **N** is reachable (i.e. can be reached by some other node) but cannot reach any other node (i.e. has no arcs leaving it).

Test your definition with the query: **deadEnd(N)**.
You should get 2 answers, e and f, one or both of which may be repeated. The repetition is fine.

ii) **hubs(H)**
to mean **H** is the set of hub nodes, i.e. those nodes N that are part of at least 3 arcs, each arc either entering N or leaving N. The result **H** should have no repetition, and should be alphabetically ordered.

Test your definition with the query: **hubs(H)**.
H should be [a,c,d,f,g].

iii) **ideal(N)**
to mean **N** is a node from which there is a path to all other nodes in the graph. You can assume that every node in the graph is connected, i.e. is the source or destination of an arc.

Test your definition with the query: **ideal(N)**.
N should be a. The answer may be repeated.

iv) **shortest_path(N1, N2, P)**
to mean **P** is a shortest path from node **N1** to node **N2**. Represent **P** as a list of nodes starting at **N1** and ending at **N2**. A path is possible only in the direction of the arrows in the graph.

Test your definition with the following queries: Some answers may be repeated. That is fine.

shortest_path(a, a, P)	Answer: **P**= [a]
shortest_path(a, e, P)	Answer: **P**= [a,b,e]
shortest_path(a, f, P)	Answers: **P** = [a,c,f] **P** = [a,d,f]
shortest_path(a, f, [a,d,c,f])	Answer: **no**
shortest_path(g, b, P)	Answer: **no**

2. Write Prolog programs for the following relations Do not use any Prolog list processing built-in functions except, if required, ***member***, ***append*** and ***length***. You can use ***setof***, ***findall,*** **\=** and the cut, ***!***. You can also use Prolog's negation operator, if needed, together with any arithmetic primitives you may need (e.g. <, >).

i) **subList(L1, L2)**

to mean every element in list **L1** is also in list **L2**, and an element in list **L1** is repeated in **L1** at most the number of times it is repeated in **L2**, and the order of the elements in list **L1** matches their order in list **L2**. You can assume the second argument **L2** is grounded in the call.

E.g.

subList([1,2,3], [5,1,1,3,2,4,3]) should succeed (in the second list there are occurrences of 1, 2, 3 in that order – it does not matter that there are other elements in between them).

subList([1,1,3,4], [5,1,3,2,3,4]) should fail (number 1 occurs twice in the first list, but only once in the second list.)

subList([1,4,3], [1,3,2,3,4]) should fail (the order of 4 and 3 in the first list does not match their order in the second list).

subList(X, [1,2,3]) should produce the answers
X = [], X = [1], X = [1,2], X = [1,2,3], X = [1,3], X = [2], X = [2,3], X = [3].
Their order is not important.

ii) **cart(L1, L2, P)**

to mean **P** is the *Cartesian product* of the lists **L1** and **L2**. *Cartesian product* is a simple concept. For example if **L1** = [a,b] and **L2**= [1,2,3], then the Cartesian product of **L1** and **L2** is the list [(a,1),(a,2),(a,3),(b,1),(b,2),(b,3)] . If either **L1** or **L2** is the empty list then their Cartesian product is also the empty list.