Graph Traversing Algorithm using prolog

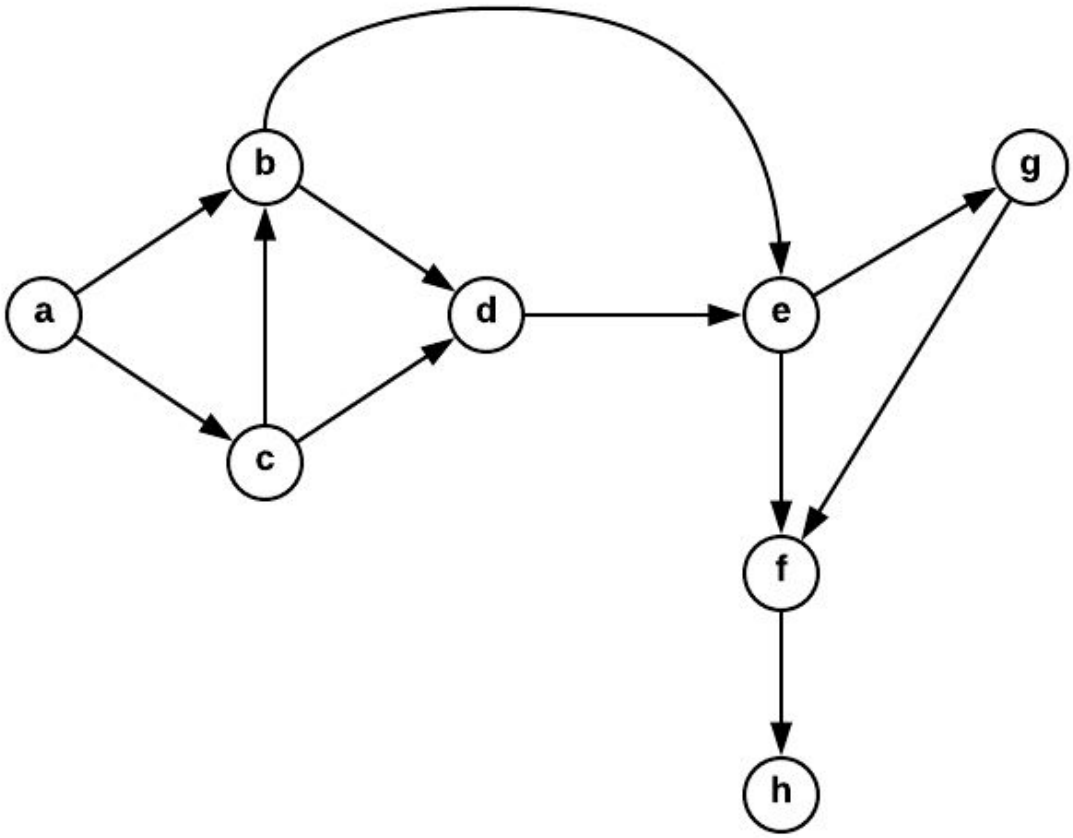

Fig : Example Directed Graph

Lest write the edge relation clause between two nodes, we will follow alphabetic order while creating such clauses.

//Start of edge clauses

edge(a,b).

edge(a,c).

edge(b,d).

edge(b,e).

edge(c,b).

edge(c,d).
edge(d,e).
edge(e,f).
edge(e,g).
edge(f,h).
edge(g,f).
//End of edge clauses

//Start of Graph traversal algorithm
//There is path exists between **X** and **Y** if there is a clause like **edge(X,Y)** exists

path(X,Y) :- edge(X,Y) , write(Y). // First Condition

//There is also a path exists between **X** and **Y** if there is a clause like **edge(X,Z) and path(Z,Y)** are true, here Z is intermediate node between X and Y. It's a recursive function, its search through all the edge and path clause until it get the path or not.

path(X,Y) :- edge(X,Z) , path(Z,Y) , write(Z). //Second Condition

Actual Trace from prolog code execution

1. Test if path exist between **a** and **g**.

```
[trace]  ?- path(a,g).
 T Call: (8) path(a, g)
 T Redo: (8) path(a, g)
 T Call: (9) path(b, g)
 T Redo: (9) path(b, g)
 T Call: (10) path(d, g)
 T Redo: (10) path(d, g)
 T Call: (11) path(e, g)
g
 T Exit: (11) path(e, g)
e
 T Exit: (10) path(d, g)
d
 T Exit: (9) path(b, g)
b
 T Exit: (8) path(a, g)
```

2. Test if path exist between **a** and **f**.

```
[trace]  ?- path(a,f).
 T Call: (8) path(a, f)
 T Redo: (8) path(a, f)
 T Call: (9) path(b, f)
 T Redo: (9) path(b, f)
 T Call: (10) path(d, f)
 T Redo: (10) path(d, f)
 T Call: (11) path(e, f)
f
 T Exit: (11) path(e, f)
e
 T Exit: (10) path(d, f)
d
 T Exit: (9) path(b, f)
b
 T Exit: (8) path(a, f)
```

We find that this above prolog graph traversal algorithm don't give shortest hop count route, this happen because we write our edge clauses in alphabetic order, so while testing clauses prolog algorithm chooses the first clause from clauses

entry. If we want that the algorithm find the shortest hop path then we should put edge(b,e) before edge(b,d).

Lets try trace prolog program execution by hand,

Our goal is to find path between a and f.

S/No	Execution Trace	Steps	X	Y	Z
1	?- path(a,f).	goal	a	f	
2	1.path(a,f) :- edge(a,f)	call_1	a	f	
3	1.path(a,f) :- false	exit_1	a	f	
4	2.path(a,f) :- edge(a,b),path(b,f)	call_2	a	f	b
5	1.path(b,f) :- edge(b,f)	call_3	b	f	
6	1.path(b,f) :- false	exit_3	b	f	
7	2.path(b,f) :- edge(b,d),path(d,f)	call_4	b	f	d
8	1.path(d,f) :- edge(d,f)	call_5	d	f	
9	1.path(d,f) :- false	exit_5	d	f	
10	2.path(d,f) :- edge(d,e),path(e,f)	call_6	d	f	e
11	1.path(e,f) :- edge(e,f)	call_7	e	f	
12	1.path(e,f) :- true,write(f)	exit_7	e	f	
13	2.path(d,f) :- edge(d,e),edge(e,f),write(e)	exit_6	d	f	e
14	2.path(b,f) :- edge(b,d),edge(d,e),write(d)	exit_4	b	e	d
15	2.path(a,f) :- edge(a,b),edge(b,d),write(b)	exit_2	a	d	b
16	?- path(a,f). true	ans			

Path would be f - e - d - b - a in reverse order.