List of programs:-

Predicates in Prolog
Construction of Family tree in Prolog
Building of Haryana Map in prolog
Source code to sort a list of unsorted list using Quick Sort in Prolog
Source code to traverse a graph using BFS in prolog
Source code to traverse a graph using DFS in prolog
Develop a Medical Expert System in Prolog
Develop a Electrical Expert System in Prolog
Source code of N-queen problem in Prolog
Source code of 8-Puzzle problem in Prolog

Program-1

Predicates in Prolog

```
domains
list=integer*
predicates
union(list,list,list)
member(integer,list)
append(list,list,list)
delete1(integer,list,list)
deleteall(integer,list,list)
occur(integer,list,integer)
replace(integer,integer,list,list)
intersection(list,list,list)
setdiff(list,list,list)
samelist(list,list)
equal(list,list)
equiv(list,list)
rev(list,list)
sum(list,integer)
palin(list)
merger(list,list,list)

clauses
member(X,[X|_]).
member(X,[H|T]):-H<>X,member(X,T).
delete1(_,[],[]).
delete1(X,[H|T1],[H|T2]):-H<>X,delete1(X,T1,T2).
delete1(X,[X|T1],T1).
deleteall(_,[],[]).
deleteall(X,[H|T1],[H|T2]):-H<>X,deleteall(X,T1,T2).
deleteall(X,[X|T1],T2):-deleteall(X,T1,T2).
union([],[],[]).
union([],L,L).
union(L,[],L).
union([H1|T1],[H2|T2],[H1|NT]):-H1=H2,union(T1,T2,NT).
union([H1|T1],[H2|T2],[H1|NT]):-H1<>H2,deleteall(H1,[H2|T2],NL),union(T1,NL,NT).
intersection([],[],[]).
intersection([],_,[]).
intersection(_,[],[]).
intersection([H1|T1],[H2|T2],[H1|NT]):-H1=H2,intersection(T1,T2,NT).
intersection([H1|T1],[H2|T2],NT):-H1<>H2,intersection(T1,[H2|T2],NT).
occur(X,[],0).
occur(X,[H|T],N):-X=H,occur(X,T,M),N=M+1.
occur(X,[H|T],N):-X<>H,occur(X,T,M),N=M.
setdiff([],[],[]).
```

```
setdiff(L,[],L).
setdiff([],_,[]).
setdiff([H1|T1],[H2|T2],NT):-H1=H2,setdiff(T1,T2,NT).
setdiff([H1|T1],[H2|T2],[H1|NT]):-H1<>H2,setdiff(T1,[H2|T2],NT).
append([],[],[]).
append(L,[],L).
append([],L,L).
append([H1|T1],L2,[H1|T3]):-append(T1,L2,T3).
rev([],[]).
rev([H|T],NL):-rev(T,NT),append(NT,[H],NL).
samelist([],[]).
samelist([H1|T1],[H2|T2]):-H1=H2,samelist(T1,T2).
equal([],[]).
equal([H1|T1],[H2|T2]):-member(H1,[H2|T2]),delete1(H1,[H2|T2],L3),equal(T1,L3).
```

Program-2

Construction of Family tree in Prolog

```
domains
person=symbol
predicates
father(person,person)
mother(person,person)
husband(person,person)
wife(person,person)
son(person,person)
daughter(person,person)
chacha(person,person)
chachi(person,person)
mama(person,person)
mami(person,person)
brother(person,person)
sister(person,person)
bua(person,person)
jija(person,person)
mausi(person,person)
male(person)
female(person)
dada(person,person)
dadi(person,person)
nana(person,person)
nani(person,person)
bhabhi(person,person)

clauses
father("prithviraj","rajkapoor").
father("prithviraj","shammi").
father("prithviraj","shashi").
father("rajkapoor","randhir").
father("rajkapoor","rishi").
father("rajkapoor","rajiv").
father("rajkapoor","ritu").
father("randhir","karishma").
father("randhir","kareena").
father("rishi","ranveer").
father("shashi","kunal").
father("shashi","sanjna").
```

```
father("rajan","nikhil").
husband("rajkapoor","krishna").
husband("shammi","geeta").
husband("shashi","jenifer").
husband("randhir","babita").
husband("rishi","neetu").
husband("rajan","ritu").
husband("sanjay","karishma").
husband("nikhil","shewta").


wife(X,Y):-husband(Y,X).
dada(X,Y):-father(X,Z),father(Z,Y).
dadi(X,Y):-wife(X,Z),dada(Z,Y).
mother(X,Y):-wife(X,Z),father(Z,Y).
son(X,Y):-father(Y,X),male(X).
daughter(X,Y):-father(Y,X),female(X).
nana(X,Y):-father(X,Z),mother(Z,Y).
nani(X,Y):-mother(X,Z),mother(Z,Y).
brother(X,Y):-father(Z,Y),father(Z,X),male(X),X<>Y.
sister(X,Y):-father(Z,Y),father(Z,X),female(X),X<>Y.
jija(X,Y):-husband(X,Z),sister(Z,Y).
bhabhi(X,Y):-wife(X,Z),brother(Z,Y).
chacha(X,Y):-brother(X,Z),father(Z,Y).
chachi(X,Y):-wife(X,Z),chacha(Z,Y).
mama(X,Y):-brother(X,Z),mother(Z,Y).
mami(X,Y):-wife(X,Z),mama(Z,Y).
mausi(X,Y):-sister(X,Z),mother(Z,Y).
bua(X,Y):-sister(X,Z),father(Z,Y).
male(X):-husband(X,_).
male("rajiv").
male("kunal").
male("ranveer").
male("prithviraj").
female(X):-wife(X,_).
female("sanjna").
female("kareena").
```

Program-3

Building of Haryana Map in prolog

```
domains
city=symbol
distance=integer
path=city*
predicates
Road(city,city,distance,path)
st_route(city,city,distance,path)
rev_route(city,city,distance,path)
append(path,path,path)
reverse(path,path)
route(city,city,distance,path)
clauses
Road("hodal","palwal",30,["hodal"]).
Road("palwal","fbd",25,["palwal"]).
Road("fbd","delhi",35,["fbd"]).
Road("delhi","sonipat",45,["delhi"]).
Road("sonipat","panipat",50,["sonipat"]).
Road("panipat","karnal",35,["panipat"]).
Road("karnal","yamunagar",60,["karnal"]).
Road("yamunagar","kkr",50,["yamunagar"]).
Road("karnal","kkr",35,["karnal"]).
Road("kkr","ambala",40,["kkr"]).
Road("ambala","chd",55,["ambala"]).
st_route(C1,C2,D,P):-Road(C1,C2,D,P).
st_route(C1,C2,D,P):-
st_route(C1,X,D1,P1),Road(X,C2,D2,P2),D=D1+D2,append(P1,P2,P).
rev_route(C1,C2,D,P):-st_route(C2,C1,D,P1),reverse(P1,P).
reverse([],[]).
reverse([H|T1],L2):-reverse(T1,L1),append(L1,[H],L2).
append([],L,L).
append(L,[],L).
append([H|T1],L2,[H|T3]):-append(T1,L2,T3).
route(C1,C2,D,P):-st_route(C1,C2,D,P).
route(C1,C2,D,P):-rev_route(C1,C2,D,P).
```

Program-4

Source code to sort a list of unsorted list using Quick Sort in Prolog

```
domains
list=integer*
predicates
qsort(list,list)
partition(integer,list,list,list)
append(list,list,list)
clauses
qsort([],[]).

qsort([X|T1],SL):-partition(X,T1,LEL,GL),qsort(LEL,SL1),qsort(GL,SL2),append(SL1,
[X|SL2],SL).
partition(_,[],[],[]).
partition(X,[H|T2],[H|T3],L4):-H<=X,partition(X,T2,T3,L4).
partition(X,[H|T2],L3,[H|T4]):-H>X,partition(X,T2,L3,T4).

append(L,[],L).
append([],L,L).
append([H|T1],L2,[H|L3]):-append(T1,L2,L3).
```

Program-5

Source code to traverse a graph using BFS in prolog

```
domains
list=symbol*
predicates
bfs(list,symbol,list)
append(list,list,list)
child(symbol,list)
clauses
child(a,[b,c,d]).
child(b,[e,f]).
child(c,[g]).
child(d,[h,i]).
child(e,[]).
child(f,[]).
child(g,[k]).
child(h,[l]).
child(i,[]).
child(k,[]).
child(l,[]).
append([],L,L).
append([H|T1],L2,[H|T3]):-append(T1,L2,T3).
bfs([],_,[]).
bfs([H|_],H,[H]).
bfs([H|T],X,[H|T3]):-H<>X,
                     child(H,L1),
                     append(T,L1,Nt),
                     bfs(Nt,X,T3).
```

Program-6

Source code to traverse a graph using DFS in prolog

```
domains
list=symbol*
predicates
dfs(list,symbol,list)
append(list,list,list)
child(symbol,list)
clauses
child(a,[b,c,d]).
child(b,[e,f]).
child(c,[g]).
child(d,[h,i]).
child(e,[]).
child(f,[]).
child(g,[k]).
child(h,[l]).
child(i,[]).
child(k,[]).
child(l,[]).
append([],L,L).
/*append(L,[],L).
append([],[],[]).*/
append([H|T1],L2,[H|T3]):-append(T1,L2,T3).
dfs([],_,[]).
dfs([H|_],H,[H]).
dfs([H|T],X,[H|T3]):-H<>X,
                     child(H,L1),
                     append(L1,T,Nt),
                     dfs(Nt,X,T3).
```

Program-7

Develop a Medical Expert System in Prolog

```
domains
sym= s1;s2;s3;s4;s5;s6
dis=d1;d2;d3;d4
med=m1;m2;m3;m4
pat=p1;p2;p3;p4
database
pat_sym_yes(pat,sym)
pat_sym_no(pat,sym)
pat_dis_yes(pat,dis)
pat_dis_no(pat,dis)
predicates
has_sym(pat,sym)
has_dis(pat,dis)
tell_med(pat)
clear

clauses
pat_sym_yes(p1,s1).
pat_sym_yes(p1,s2).
pat_sym_yes(p2,s2).
pat_sym_yes(p3,s4).
pat_sym_yes(p4,s5).
pat_sym_yes(p4,s2).
has_sym(P,S):-pat_sym_yes(P,S),!.
has_sym(P,S):-pat_sym_no(P,S),!,fail.
has_sym(P,S):-write("\n Mr " ,P,"do u have
symtom",S ,"(y/n)?"),readln(Ans),Ans="y",assert(pat_sym_yes(P,S));assert(pat_sym_no(
P,S)).
has_dis(P,d1):-pat_dis_yes(P,d1),!.
has_dis(P,d1):-pat_dis_no(P,d1),!,fail.
has_dis(P,d1):-has_sym(P,s1),has_sym(P,s3),has_sym(P,s5).
has_dis(P,d2):-pat_dis_yes(P,d2),!.
has_dis(P,d2):-pat_dis_no(P,d2),!,fail.
has_dis(P,d2):-has_sym(P,s2),has_sym(P,s4),has_sym(P,s5).
has_dis(P,d3):-pat_dis_yes(P,d3),!.
has_dis(P,d3):-pat_dis_no(P,d3),!,fail.
has_dis(P,d3):-has_sym(P,s2),has_sym(P,s3),has_sym(P,s5).
has_dis(P,d4):-pat_dis_yes(P,d4),!.
has_dis(P,d4):-pat_dis_no(P,d4),!,fail.
has_dis(P,d4):-has_sym(P,s1),has_sym(P,s4),has_sym(P,s5).
tell_med(P):-has_dis(P,d1),write("\n Mr",P,"do take medicine",m1).
```

```
tell_med(P):-has_dis(P,d2),write("\n Mr",P,"do take medicine",m2).
tell_med(P):-has_dis(P,d3),write("\n Mr",P,"do take medicine",m3).
tell_med(P):-has_dis(P,d4),write("\n Mr",P,"do take medicine",m4).
clear:-retract(pat_sym_yes(_,_)),fail.
clear:-retract(pat_sym_no(_,_)),fail.
clear:-retract(pat_dis_yes(_,_)),fail.
clear:-retract(pat_dis_no(_,_)),fail.

clear.
```

Program-10

Source code of 8-Puzzle problem in Prolog

```
domains
list=integer*
listoflist=list*
predicates
children(list,listoflist)
bfs(listoflist,list,listoflist)
append(listoflist,listoflist,listoflist)
samelist(list,list)
clauses
samelist([X],[X]).
samelist([H1|T1],[H2|T2]):-H1=H2,samelist(T1,T2).
children([0,A,B,C,D,E,F,G,H],[[A,0,B,C,D,E,F,G,H],[C,A,B,0,D,E,F,G,H]]).
children([A,0,B,C,D,E,F,G,H],[[0,A,B,C,D,E,F,G,H],[A,B,0,C,D,E,F,G,H],
[A,D,B,C,0,E,F,G,H]]).
children([A,B,0,C,D,E,F,G,H],[[A,0,B,C,D,E,F,G,H],[A,B,E,C,D,0,F,G,H]]).
children([A,B,C,0,D,E,F,G,H],[[0,B,C,A,D,E,F,G,H],[A,B,C,F,D,E,0,G,H],
[A,B,C,D,0,E,F,G,H]]).
children([A,B,C,D,0,E,F,G,H],[[A,0,C,D,B,E,F,G,H],[A,B,C,D,G,E,F,0,H],
[A,B,C,0,D,E,F,G,H],[A,B,C,D,E,0,F,G,H]]).
children([A,B,C,D,E,0,F,G,H],[[A,B,0,D,E,C,F,G,H],[A,B,C,D,E,H,F,G,0],
[A,B,C,D,0,E,F,G,H]]).
children([A,B,C,D,E,F,0,G,H],[[A,B,C,0,E,F,D,G,H],[A,B,C,D,E,F,G,0,H]]).
children([A,B,C,D,E,F,G,0,H],[[A,B,C,D,0,F,G,E,H],[A,B,C,D,E,F,0,G,H],
[A,B,C,D,E,F,G,H,0]]).
children([A,B,C,D,E,F,G,H,0],[[A,B,C,D,E,F,G,0,H],[A,B,C,D,E,0,G,H,F]]).
append([],[L],[L]).
append([[H]|T1],[L2],[[H]|T3]):-append(T1,[L2],T3).
/*bfs([],_,[]).
bfs([H|_],[A,B,C,D,E,F,G,H,0],[H]).
bfs([H|T],[A,B,C,D,E,F,G,H,0],[H|T3]):-H<>X,
                    children([H],L1),
                    append(T,L1,Nt),
                    bfs(Nt,X,T3).*/
bfs([L1|_],L2,[L1]):-samelist(L1,L2),!.
bfs([L1|T1],L2,[Path]):-
not(samelist(L1,L2)),children(L1,L3),append(T1,L3,Nlist),bfs(Nlist,L2,
[Npath]),append([L1],[Npath],[Path]).
```