Name: Brijesh Rameshbhai Rohit

Admission number: U19CS009

AI-ASSIGNMENT-09

Implement 8 Puzzle problem using below algorithms in prolog. Compare the complexity of both algorithms. Which algorithm is best suited for implementing 8 Puzzle problem and why?

1. DFS

Code:

```
hswap([A1,A2,A3],1,[A2,A1,A3]).
hswap([A1,A2,A3],2,[A1,A3,A2]).

vswap([A1,A2,A3],[B1,B2,B3],1,[B1,A2,A3],[A1,B2,B3]).
vswap([A1,A2,A3],[B1,B2,B3],2,[A1,B2,A3],[B1,A2,B3]).
vswap([A1,A2,A3],[B1,B2,B3],3,[A1,A2,B3],[B1,B2,A3]).

goal([[1,2,3],[4,5,6],[7,8,w]]).

getcells([H1,H2,H3],[H1,H2,H3]).

dfs(Path,Node,Path,_,_,_):-
    goal(Node).


%RIGHT
dfs(Path,Node,Sol,I,J,K):-
    I>0,I=<3,J>0,J<3,
    % K<5,K1 is K+1,
    % write(Node),nl,
    getcells(Node,[A,B,C]),
    (
        (I==1)->(
            hswap(A,J,X),not(member([X,B,C],Path)),J1 is
J+1,dfs([[X,B,C]|Path],[X,B,C],Sol,I,J1,K)
            );
        (I==2)->(
            hswap(B,J,X),not(member([A,X,C],Path)),J1 is
J+1,dfs([[A,X,C]|Path],[A,X,C],Sol,I,J1,K)
            );
        (I==3)->(
```

```
            hswap(C,J,X),not(member([A,B,X],Path)),J1 is
J+1,dfs([[A,B,X]|Path],[A,B,X],Sol,I,J1,K)
            )

        ),!.
%DOWN
dfs(Path,Node,Sol,I,J,K):-
    I>0,I<3,J>0,J=<3,
    % K<5,K1 is K+1,
    % write(Node),nl,
    getcells(Node,[A,B,C]),
    (
        (I==2)->(
        vswap(B,C,J,X,Y),not(member([A,X,Y], Path)),I1 is
I+1,dfs([[A,X,Y]|Path],[A,X,Y],Sol,I1,J,K)
            );
        (I==1)->(
            vswap(A,B,J,X,Y),not(member([X,Y,C], Path)),I1 is
I+1,dfs([[X,Y,C]|Path],[X,Y,C],Sol,I1,J,K)
            )
    ),!.
%UP
dfs(Path,Node,Sol,I,J,K):-
    I>1,I=<3,J>0,J=<3,
    % K<5,K1 is K+1,
    % write(Node),nl,
    getcells(Node,[A,B,C]),
    (
        (I==2)->(
        vswap(A,B,J,X,Y),not(member([X,Y,C], Path)),I1 is
I-1,dfs([[X,Y,C]|Path],[X,Y,C],Sol,I1,J,K)
            );
        (I==3)->(
            vswap(B,C,J,X,Y),not(member([A,X,Y], Path)),I1 is
I-1,dfs([[A,X,Y]|Path],[A,X,Y],Sol,I1,J,K)
            )
    ),!.


%LEFT
dfs(Path,Node,Sol,I,J,K):-
    I>0,I=<3,J>1,J=<3,
    % K<5,K1 is K+1,
    % write(Node),nl,
    getcells(Node,[A,B,C]),
    (
```

```
        J1 is J-1,
        (I==1)->(

hswap(A,J1,X),not(member([X,B,C],Path)),dfs([[X,B,C]|Path],[X,B,C],So
l,I,J1,K)
            );
        (I==2)->(

hswap(B,J1,X),not(member([A,X,C],Path)),dfs([[A,X,C]|Path],[A,X,C],So
l,I,J1,K)
            );
        (I==3)->(

hswap(C,J1,X),not(member([A,B,X],Path)),dfs([[A,B,X]|Path],[A,B,X],So
l,I,J1,K)
            )

        ),!.

print([A,B,C]):-
    write(A),nl,write(B),nl,write(C),nl,nl.

display([]).
display([H|T]):-
    display(T),
    print(H).
solve(Node,I,J):-
    dfs([],Node,Sol,I,J,0),print(Node),display(Sol).
```

Output:

```
5 ?- solve([[1,2,3],[4,w,5],[7,8,6]],2,2).
[1,2,3]
[4,w,5]
[7,8,6]

[1,2,3]
[4,5,w]
[7,8,6]

[1,2,3]
[4,5,6]
[7,8,w]

true.
```

2. BFS

Code:

```
%U19CS009
%BRIJESH ROHIT

%PREDICATE TO PERFORM HORIZONAL SWAP
hswap([A1,A2,A3],1,[A2,A1,A3]).
hswap([A1,A2,A3],2,[A1,A3,A2]).

%PREDICATE TO PERFORM VERTICAL SWAP
vswap([A1,A2,A3],[B1,B2,B3],1,[B1,A2,A3],[A1,B2,B3]).
vswap([A1,A2,A3],[B1,B2,B3],2,[A1,B2,A3],[B1,A2,B3]).
vswap([A1,A2,A3],[B1,B2,B3],3,[A1,A2,B3],[B1,B2,A3]).

%FINAL GOAL STATE
goal([[1,2,3],[4,5,6],[7,8,w],[3,3]]).

%PREDICATE TO CHECK IF VALUES ARE SAME OR NOT
sets(X,X).

%PREDICATE TO GET RIGHT POSSIBILITY
```

```
getright([A,B,C,[I,J]],Right):-
    I>0,I=<3,J>0,J<3,
    (
        (I==1)->
            hswap(A,J,X),J1 is J+1,sets([X,B,C,[I,J1]],Right);
        (I==2)->
            hswap(B,J,X),J1 is J+1,sets([A,X,C,[I,J1]],Right);
        (I==3)->
            hswap(C,J,X),J1 is J+1,sets([A,B,X,[I,J1]],Right)
    ).
getright(_,[]).

%PREDICATE TO GET LEFT POSSIBILITY
getleft([A,B,C,[I,J]],Left):-
    I>0,I=<3,J>1,J=<3,
    J1 is J-1,
    (
        (I==1)->
            hswap(A,J1,X),sets([X,B,C,[I,J1]],Left);
        (I==2)->
            hswap(B,J1,X),sets([A,X,C,[I,J1]],Left);
        (I==3)->
            hswap(C,J1,X),sets([A,B,X,[I,J1]],Left)
    ).

getleft(_,[]).

%PREDICATE TO GET UP POSSIBILITY
getup([A,B,C,[I,J]],Up):-
    I>1,I=<3,J>0,J=<3,
    (
        (I==2)->
            vswap(A,B,J,X,Y),I1 is I-1,sets(Up,[X,Y,C,[I1,J]]);
        (I==3)->
            vswap(B,C,J,X,Y),I1 is I-1,sets(Up,[A,X,Y,[I1,J]])
    ),!.

getup(_,[]).

%PREDICATE TO GET DOWN POSSIBILITY
getdown([A,B,C,[I,J]],Down):-
    (
        (I==2)->
        vswap(B,C,J,X,Y),I1 is I+1,sets([A,X,Y,[I1,J]],Down);
        (I==1)->
            vswap(A,B,J,X,Y),I1 is I+1,sets([X,Y,C,[I1,J]],Down)
```

```
    ),!.

getdown(_,[]).

%PREDICATE TO APPEND
append([],X,X).
append([H|T],N,[H|T1]):-
    append(T,N,T1).

%PREICATE TO PRINT THE SOLUTION
print([A,B,C,_]):-
    write(A),nl,write(B),nl,write(C),nl,nl.
display([]).
display([H|T]):-
    display(T),
    print(H).


%TO GET LIST OF ADJACENT NODES OF THE CURRENT NODE
extend([Node|Path],NewPaths):-
    bagof(
        [NewNode,Node|Path],
            (
                (

(getup(Node,X),sets(X,NewNode),not(sets(NewNode,[]))));

(getdown(Node,X),sets(X,NewNode),not(sets(NewNode,[]))));

(getleft(Node,X),sets(X,NewNode),not(sets(NewNode,[]))));

(getright(Node,X),sets(X,NewNode),not(sets(NewNode,[])))
                ),
                not(member(NewNode,[Node|Path]))
            ),NewPaths
        ),!.
extend(_,[]).

%PERFORMING BFS
bfs([[Node|Path]|_],[Node|Path],_):-
    goal(Node),write("\ndone\n"),!.
bfs([Path|Paths],Sol,K):-
    K<3,K1 is K+1,
    extend(Path,NewPath),
    append(Paths,NewPath,Path1),
    bfs(Path1,Sol,K1).
```

```
%SOLVING FUNCTION
solve(Node):-
    bfs([[Node]],X,0)
    ,display(X).
```

Output:

```
1 ?- consult('bfs.pl').
true.

2 ?- solve([[1,w,2],[4,5,3],[7,8,6],[1,2]]).

done
[1,w,2]
[4,5,3]
[7,8,6]

[1,2,w]
[4,5,3]
[7,8,6]

[1,2,3]
[4,5,w]
[7,8,6]

[1,2,3]
[4,5,6]
[7,8,w]

true.
```

3. Uniform Cost Search

Since cost of each step in this 8-puzzle problem is same, Uniform cost search will be same as breadth first search.

CONCLUSION:
Time-Complexity:
Depth First Search: O(bM).
Breadth First Search: O(b_D).
Uniform Cost Search: O(b_D).
b: Number of Branches. (On average for 3x3 matrix it is 3).
M: Maximum depth of the tree
D: Depth at which goal state lies.
If time and space complexity are not the issue then BFS is better as it would return shortest path to our goal state which was not possible with DFS (DFS would have returned the 1st possible path to goal state rather than the shortest one).