Artificial Intelligence

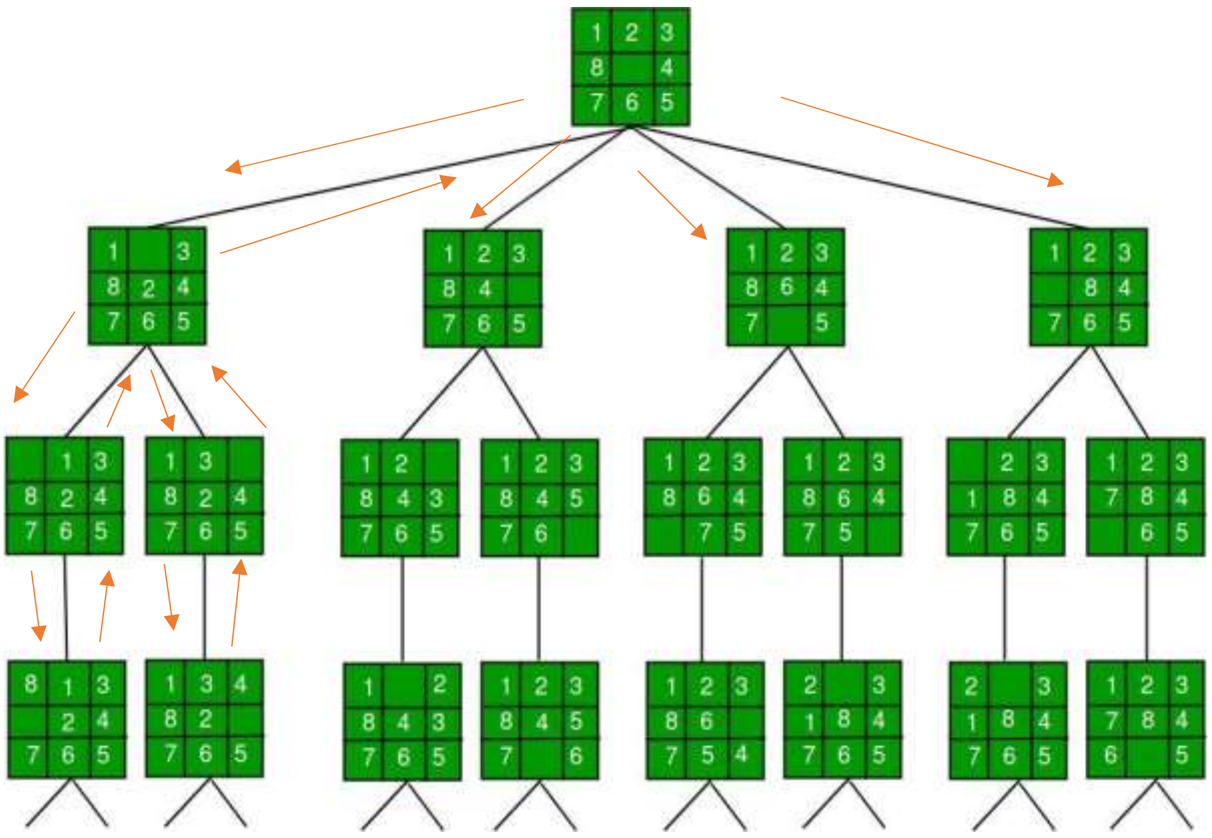
Lab Assignment

NAME: SOURABH PATEL

ADMISSION NO: U19CS082

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. . **Depth First Search**

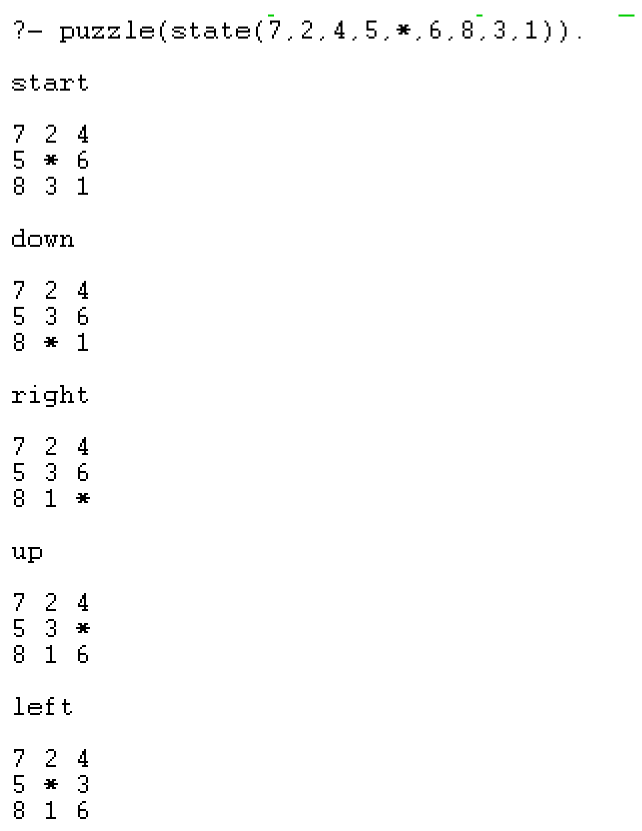


Code:-

```
puzzle(State):-
   length(Moves, N),
   dfs([State], Moves, Path), !,
   show([start|Moves], Path),
   format('~nmoves = ~w~n', [N]).

dfs([State|States], [], Path) :-
   goal(State), !,
   reverse([State|States], Path).
```

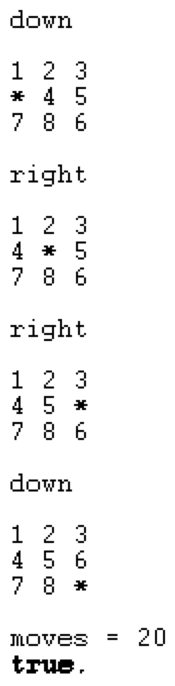
```
dfs([State|States], [Move|Moves], Path) :-
   move(State, Next, Move),
   not(memberchk(Next, [State|States])),
   dfs([Next,State|States], Moves, Path).

show([], _).
show([Move|Moves], [State|States]) :-
   State = state(A,B,C,D,E,F,G,H,I),
   format('~n~w~n~n', [Move]),
   format('~w ~w ~w~n',[A,B,C]),
   format('~w ~w ~w~n',[D,E,F]),
   format('~w ~w ~w~n',[G,H,I]),
   show(Moves, States).

% Empty position is marked with '*'



goal( state(1,2,3,4,5,6,7,8,*) ).

move( state(*,B,C,D,E,F,G,H,J), state(B,*,C,D,E,F,G,H,J), right).
move( state(*,B,C,D,E,F,G,H,J), state(D,B,C,*,E,F,G,H,J), down ).
move( state(A,*,C,D,E,F,G,H,J), state(*,A,C,D,E,F,G,H,J), left ).
move( state(A,*,C,D,E,F,G,H,J), state(A,C,*,D,E,F,G,H,J), right).
move( state(A,*,C,D,E,F,G,H,J), state(A,E,C,D,*,F,G,H,J), down ).
move( state(A,B,*,D,E,F,G,H,J), state(A,*,B,D,E,F,G,H,J), left ).
move( state(A,B,*,D,E,F,G,H,J), state(A,B,F,D,E,*,G,H,J), down ).
move( state(A,B,C,*,E,F,G,H,J), state(*,B,C,A,E,F,G,H,J), up   ).
move( state(A,B,C,*,E,F,G,H,J), state(A,B,C,E,*,F,G,H,J), right).
move( state(A,B,C,*,E,F,G,H,J), state(A,B,C,G,E,F,*,H,J), down ).
move( state(A,B,C,D,*,F,G,H,J), state(A,*,C,D,B,F,G,H,J), up   ).
move( state(A,B,C,D,*,F,G,H,J), state(A,B,C,D,F,*,G,H,J), right).
move( state(A,B,C,D,*,F,G,H,J), state(A,B,C,D,H,F,G,*,J), down ).
move( state(A,B,C,D,*,F,G,H,J), state(A,B,C,*,D,F,G,H,J), left ).
move( state(A,B,C,D,E,*,G,H,J), state(A,B,*,D,E,C,G,H,J), up   ).
move( state(A,B,C,D,E,*,G,H,J), state(A,B,C,D,*,E,G,H,J), left ).
move( state(A,B,C,D,E,*,G,H,J), state(A,B,C,D,E,J,G,H,*), down ).
move( state(A,B,C,D,E,F,*,H,J), state(A,B,C,D,E,F,H,*,J), left ).
move( state(A,B,C,D,E,F,*,H,J), state(A,B,C,*,E,F,D,H,J), up   ).
move( state(A,B,C,D,E,F,G,*,J), state(A,B,C,D,E,F,*,G,J), left ).
move( state(A,B,C,D,E,F,G,*,J), state(A,B,C,D,*,F,G,E,J), up   ).
move( state(A,B,C,D,E,F,G,*,J), state(A,B,C,D,E,F,G,J,*), right).
move( state(A,B,C,D,E,F,G,H,*), state(A,B,C,D,E,*,G,H,F), up   ).
move( state(A,B,C,D,E,F,G,H,*), state(A,B,C,D,E,F,G,*,H), left ).
```

Output:-



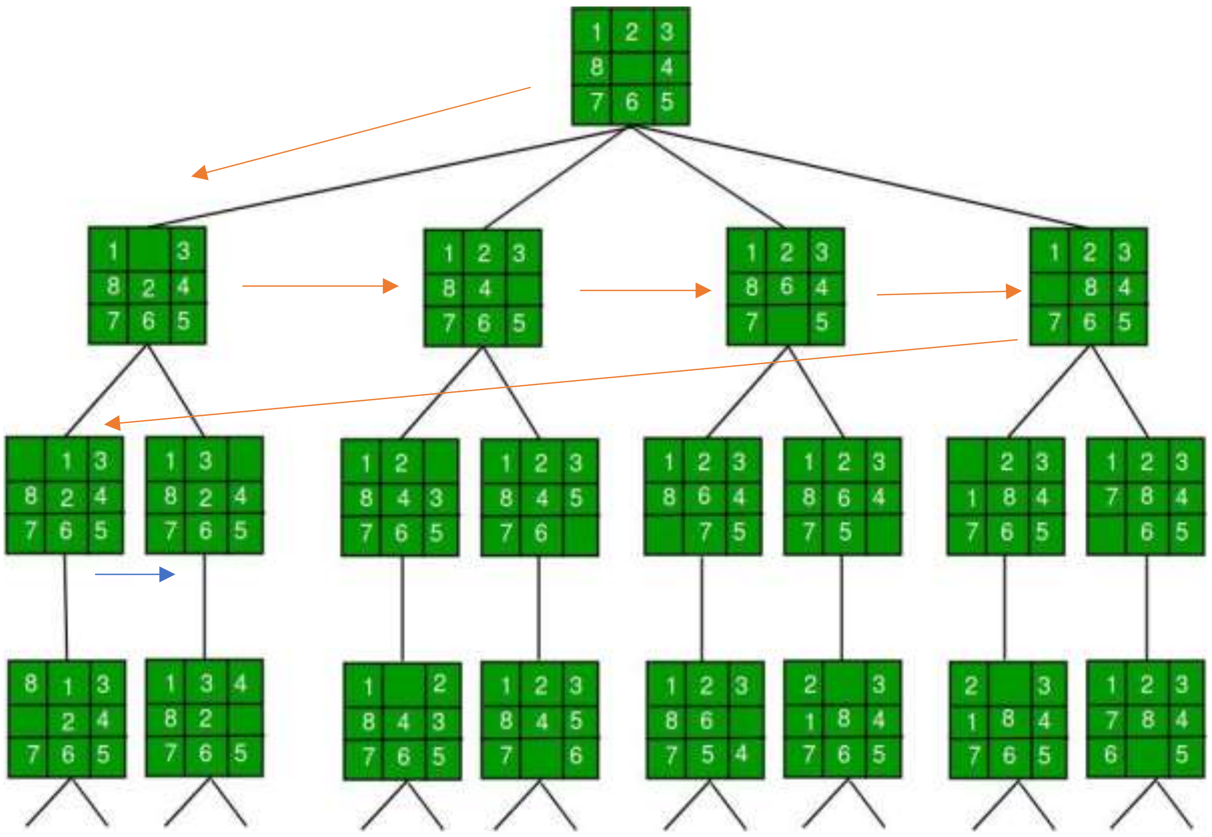
```
?- puzzle(state(7,2,4,5,*,6,8,3,1)).

start

7 2 4
5 * 6
8 3 1

down

7 2 4
5 3 6
8 * 1

right

7 2 4
5 3 6
8 1 *

up

7 2 4
5 3 *
8 1 6

left

7 2 4
5 * 3
8 1 6
```

.....

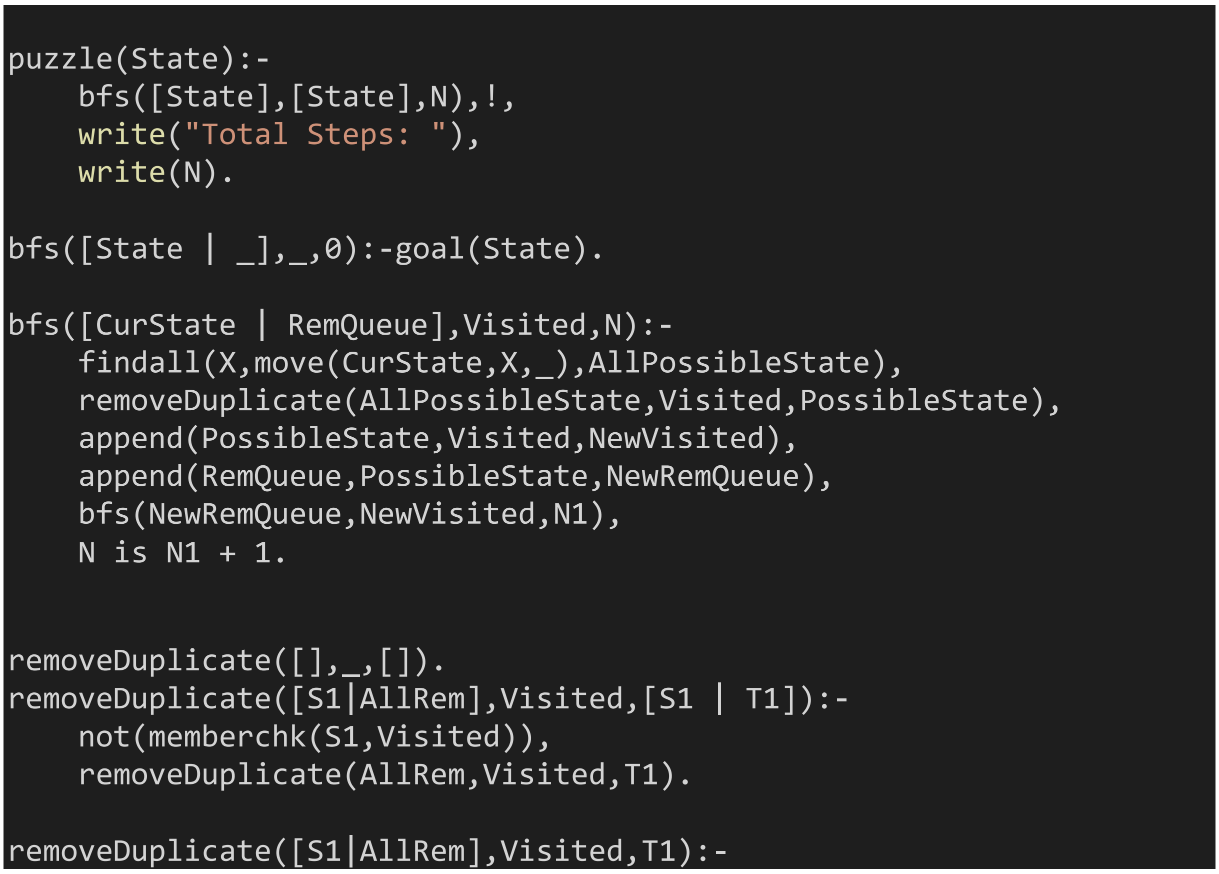


```
down

1 2 3
* 4 5
7 8 6

right

1 2 3
4 * 5
7 8 6

right

1 2 3
4 5 *
7 8 6

down

1 2 3
4 5 6
7 8 *

moves = 20
true.
```

2. Breadth First Search



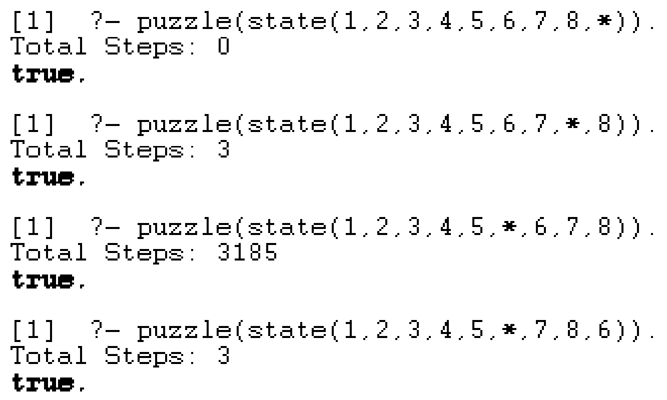
Code:



```
puzzle(State):-
    bfs([State],[State],N),!,
    write("Total Steps: "),
    write(N).

bfs([State | _],_,0):-goal(State).

bfs([CurState | RemQueue],Visited,N):-
    findall(X,move(CurState,X,_),AllPossibleState),
    removeDuplicate(AllPossibleState,Visited,PossibleState),
    append(PossibleState,Visited,NewVisited),
    append(RemQueue,PossibleState,NewRemQueue),
    bfs(NewRemQueue,NewVisited,N1),
    N is N1 + 1.


removeDuplicate([],_,[]).
removeDuplicate([S1|AllRem],Visited,[S1 | T1]):-
    not(memberchk(S1,Visited)),
    removeDuplicate(AllRem,Visited,T1).

removeDuplicate([S1|AllRem],Visited,T1):-
```

```
    memberchk(S1,Visited),
    removeDuplicate(AllRem,Visited,T1).


goal( state(1,2,3,4,5,6,7,8,*) ).

move( state(*,B,C,D,E,F,G,H,J), state(B,*,C,D,E,F,G,H,J), right).
move( state(*,B,C,D,E,F,G,H,J), state(D,B,C,*,E,F,G,H,J), down ).
move( state(A,*,C,D,E,F,G,H,J), state(*,A,C,D,E,F,G,H,J), left ).
move( state(A,*,C,D,E,F,G,H,J), state(A,C,*,D,E,F,G,H,J), right).
move( state(A,*,C,D,E,F,G,H,J), state(A,E,C,D,*,F,G,H,J), down ).
move( state(A,B,*,D,E,F,G,H,J), state(A,*,B,D,E,F,G,H,J), left ).
move( state(A,B,*,D,E,F,G,H,J), state(A,B,F,D,E,*,G,H,J), down ).
move( state(A,B,C,*,E,F,G,H,J), state(*,B,C,A,E,F,G,H,J), up   ).
move( state(A,B,C,*,E,F,G,H,J), state(A,B,C,E,*,F,G,H,J), right).
move( state(A,B,C,*,E,F,G,H,J), state(A,B,C,G,E,F,*,H,J), down ).
move( state(A,B,C,D,*,F,G,H,J), state(A,*,C,D,B,F,G,H,J), up   ).
move( state(A,B,C,D,*,F,G,H,J), state(A,B,C,D,F,*,G,H,J), right).
move( state(A,B,C,D,*,F,G,H,J), state(A,B,C,D,H,F,G,*,J), down ).
move( state(A,B,C,D,*,F,G,H,J), state(A,B,C,*,D,F,G,H,J), left ).
move( state(A,B,C,D,E,*,G,H,J), state(A,B,*,D,E,C,G,H,J), up   ).
move( state(A,B,C,D,E,*,G,H,J), state(A,B,C,D,*,E,G,H,J), left ).
move( state(A,B,C,D,E,*,G,H,J), state(A,B,C,D,E,J,G,H,*), down ).
move( state(A,B,C,D,E,F,*,H,J), state(A,B,C,D,E,F,H,*,J), left ).
move( state(A,B,C,D,E,F,*,H,J), state(A,B,C,*,E,F,D,H,J), up   ).
move( state(A,B,C,D,E,F,G,*,J), state(A,B,C,D,E,F,*,G,J), left ).
move( state(A,B,C,D,E,F,G,*,J), state(A,B,C,D,*,F,G,E,J), up   ).
move( state(A,B,C,D,E,F,G,*,J), state(A,B,C,D,E,F,G,J,*), right).
move( state(A,B,C,D,E,F,G,H,*), state(A,B,C,D,E,*,G,H,F), up   ).
move( state(A,B,C,D,E,F,G,H,*), state(A,B,C,D,E,F,G,*,H), left ).
```

Output:



```
[1]  ?- puzzle(state(1,2,3,4,5,6,7,8,*)).
Total Steps: 0
true.

[1]  ?- puzzle(state(1,2,3,4,5,6,7,*,8)).
Total Steps: 3
true.

[1]  ?- puzzle(state(1,2,3,4,5,*,6,7,8)).
Total Steps: 3185
true.

[1]  ?- puzzle(state(1,2,3,4,5,*,7,8,6)).
Total Steps: 3
true.
```

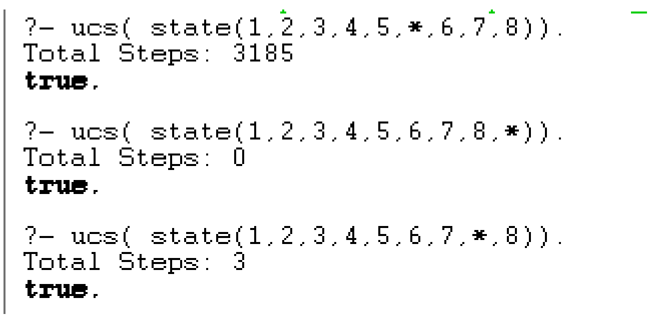
3. Uniform Cost Search

As the cost of the step is same for all steps uniform cost search will work same as breadth first search.

Code:

```
ucs(State):-
    bfs([State],[State],N),!,
    write("Total Steps: "),
    write(N).
```

Output:



```
?- ucs( state(1,2,3,4,5,*,6,7,8)).
Total Steps: 3185
true.

?- ucs( state(1,2,3,4,5,6,7,8,*)).
Total Steps: 0
true.

?- ucs( state(1,2,3,4,5,6,7,*,8)).
Total Steps: 3
true.
```

Time Complexity

1.) Depth First Search – **O(b^m)**
2.) Breadth First Search – **O(b^d)**
3.) Uniform Cost Search – **O(b^d)**
    b- number of branches = 3(average)
    d- depth
    m- max depth of tree

Which Algorithm is better?

It depends on the goal state. If goal state is not deeper in the space search tree then BFS is optimal otherwise if goal state is deeper in space search tree then DFS is better but in general

Time taken by DFS depends on the depth of the entire search tree (which could be infinite, if loopy paths are not eliminated). Even in the case of a search tree with finite depth, the **time complexity** will be O(b^*m*), where *m* is the maximum depth of the search tree, which could be much larger than *d* (the depth of the shallowest goal).

The main reason for the use of DFS over BFS in many areas of AI is because of its very less space consumption. In DFS, we need to store only the nodes which are present in the path from the root to the current node and their unexplored successors. For state space with branching factor *b* and maximum depth *m*, DFS has **space complexity** of O(*bm*), a much better improvement over that of BFS.

Submitted By:
U19CS082
Sourabh Patel