Artificial Intelligence (CS308)

Assignment - 9

**U19CS012**

Q.) Implement **8 Puzzle** problem using below algorithms in prolog.

1.) Breadth First Search

* **Shallowest unexpanded** node is chosen for expansion
* Store frontier of nodes in **FIFO** queue
* Check if **goal** when generated, since placed on Queue and taken off of queue in same order
* Check to **avoid repeated** states

**Code**

*% Breadth For Search Implementation of the 8 Puzzle Problem*

*% There are Cases When the Stack Overflows (>1 GB), Due to Space Complexity of BFS O(b^d)*

*% Main F(x) to Solve the 8 Puzzle Problem*

puzzle(State)*:-*

    bfs([State],[State],N),*!*,

    write("Total Steps: "),

    write(N)*.*

*% Check if the Goal is Reached*

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

bfs([CurState | RemQueue],Visited,N)*:-*

*% Find all the Possible Moves*

    findall(X,move(CurState,X,\_),AllPossibleState),

*% Avoid Repeated Nodes*

    removeDuplicate(AllPossibleState,Visited,PossibleState),

*% Store Frontier of Nodes in FIFO Queue*

    append(PossibleState,Visited,NewVisited),

*% Store Remaining in Other Queue*

    append(RemQueue,PossibleState,NewRemQueue),

*% Recur for Updated Queue with New Visited Nodes*

    bfs(NewRemQueue,NewVisited,N1),

*% Count the Required Number of Steps*

    N is N1 + 1*.*

*% F(x) to Remove Duplicates from the List*

*% Base Case*

removeDuplicate([],\_,[])*.*

*% If Member is Not in the List, then Add it to Answer*

removeDuplicate([H1|AllRem],Visited,[H1 | T1])*:-*

    not(memberchk(H1,Visited)),

    removeDuplicate(AllRem,Visited,T1)*.*

*% If Member is in the List, then Don't Add it to Answer*

removeDuplicate([H1|AllRem],Visited,T1)*:-*

    memberchk(H1,Visited),

    removeDuplicate(AllRem,Visited,T1)*.*

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

*% Reference for Below Moves - https://stackoverflow.com/questions/67642302/prolog-for-eight-puzzle*

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 )*.*

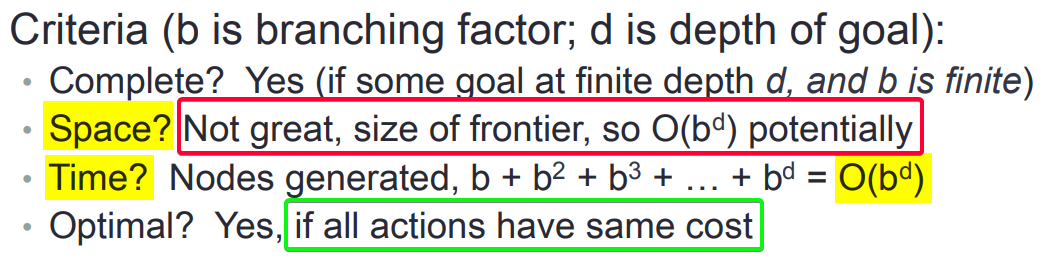
move( state(A,B,C,D,E,F,G,\*,J), state(A,B,C,D,\*,F,G,E,J), up   )*.*

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 )*.*

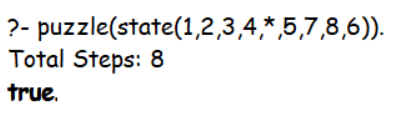
**Analysis**



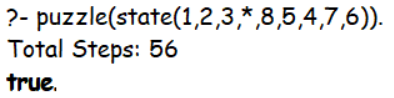
Note: **Space Complexity** is Major Problem in **Breadth for Search** Approach.

**Output**

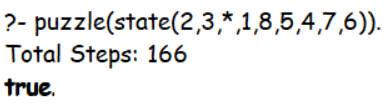
**Easy**: puzzle(state(1,2,3,4,\*,5,7,8,6)).



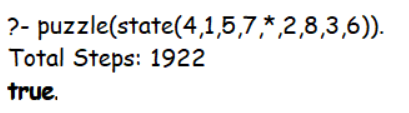
**Medium**: puzzle(state(1,2,3,\*,8,5,4,7,6)).



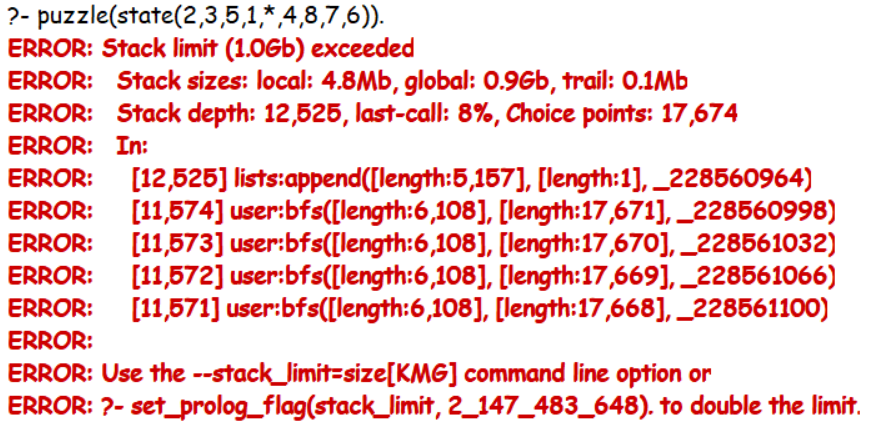
**Hard**: puzzle(state(2,3,\*,1,8,5,4,7,6)).

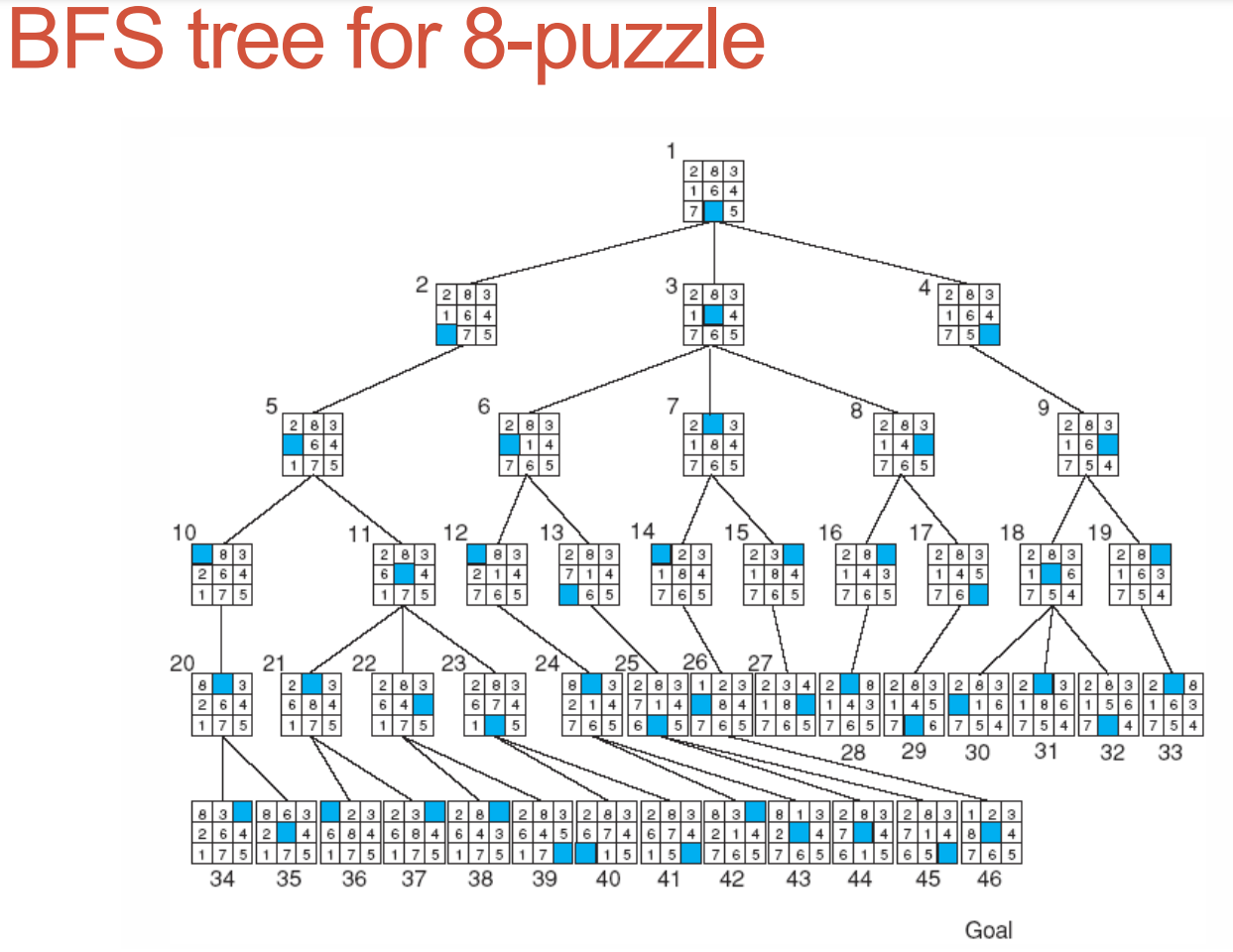


**Hardest** (Computable): puzzle(state(4,1,5,7,\*,2,8,3,6)).



**Hardest** (Not Computable): puzzle(state(2,3,5,1,\*,4,8,7,6)).





2.) Depth First Search

* Always expand the **Deepest** **Node** in the **Current** Frontier
* Uses a **LIFO** queue (a.k.a. **Stack**)
* Commonly Implemented with **Recursion**

**Code**

*% Depth For Search Implementation of the 8 Puzzle Problem*

*% Main F(x) to Solve the 8 Puzzle Problem*

puzzle(State)*:-*

    length(Moves, N),

    dfs([State], Moves, Path), *!*,

    show([start|Moves], Path),

    format('~nmoves = ~w~n', [N])*.*

dfs([State|States], [], Path) *:-*

*% If the Goal State is Reached*

    goal(State), *!*,

*%  Otherwise Backtrack using Stack*

    reverse([State|States], Path)*.*

dfs([State|States], [Move|Moves], Path) *:-*

*% Move to Deepest State Possible*

    move(State, Next, Move),

*% If the 'Next' is Not a Member of States*

    not(memberchk(Next, [State|States])),

*% Recur for Next State*

    dfs([Next,State|States], Moves, Path)*.*

*% F(x) to Display the 8 Puzzle Board*

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)*.*

*% Goal State to be Acheived*

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

*% Empty position is marked with '\*'*

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   )*.*

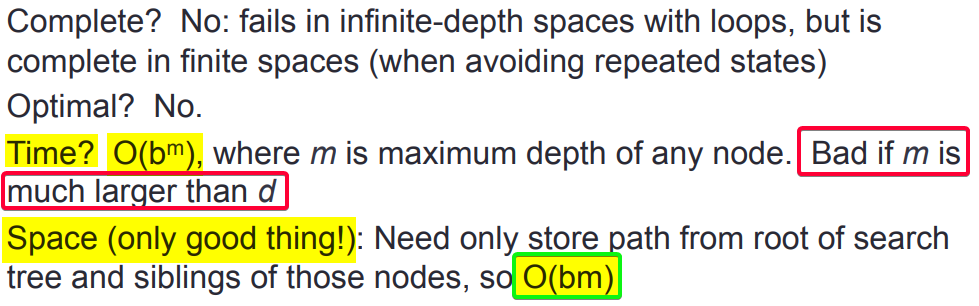
move( state(A,B,C,D,E,F,G,\*,J), state(A,B,C,D,E,F,G,J,\*), right)*.*

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 )*.*

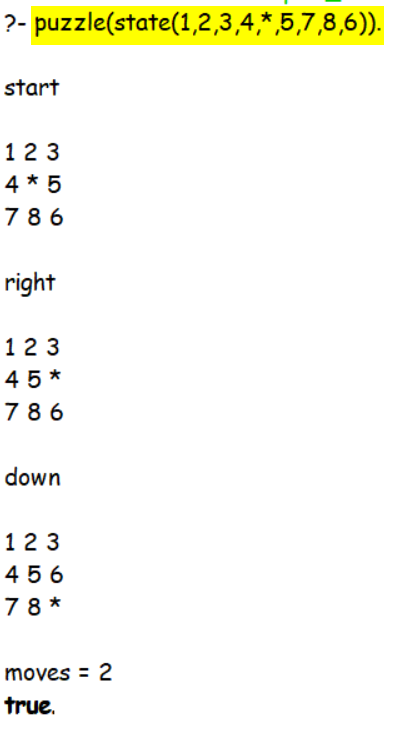
*% Reference for Code - https://stackoverflow.com/questions/67642302/prolog-for-eight-puzzle*

**Analysis**

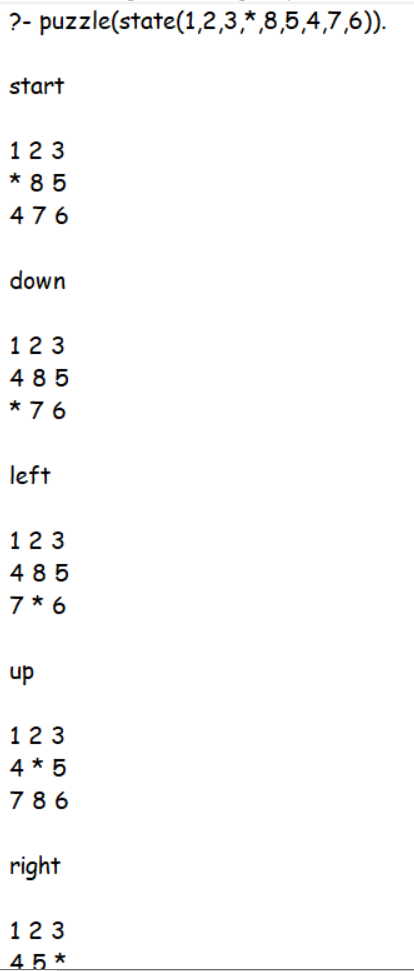
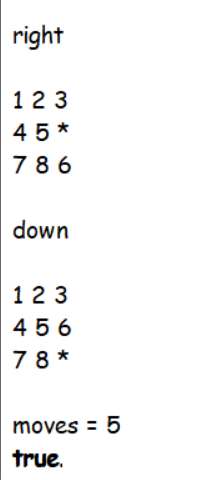


**Output**

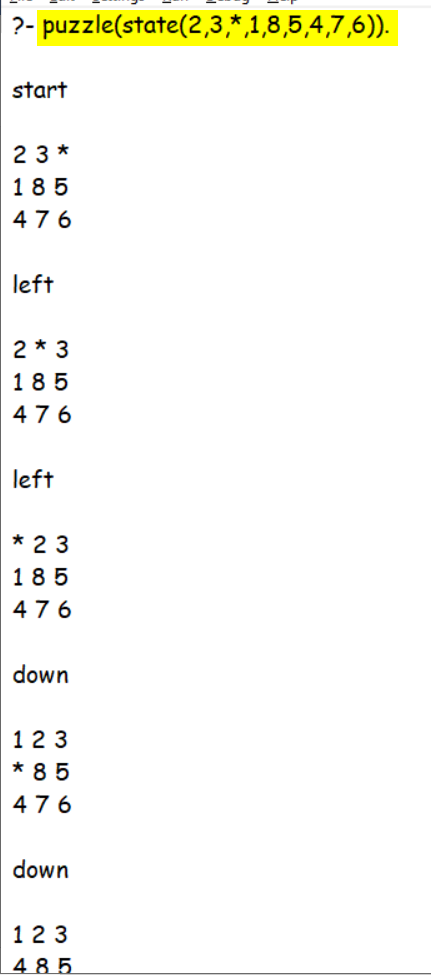
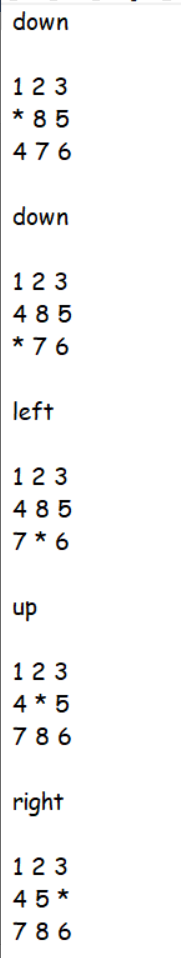
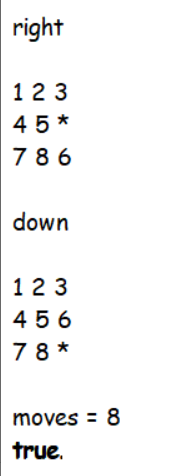
**Easy**: puzzle(state(1,2,3,4,\*,5,7,8,6)).



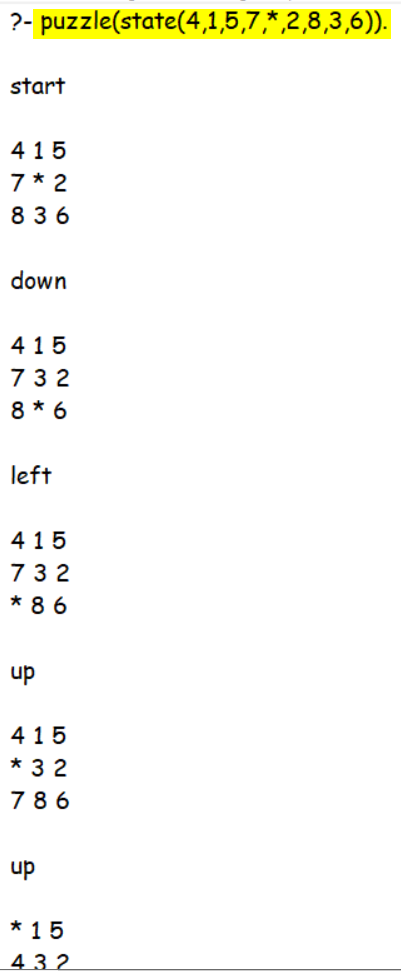
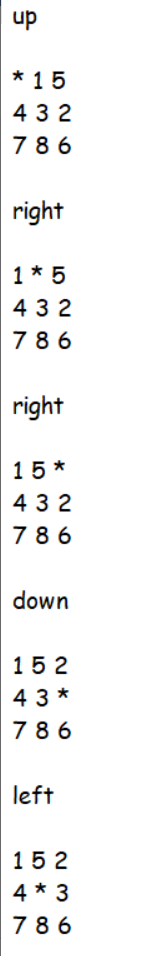
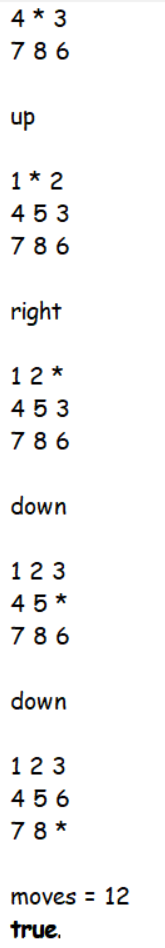
**Medium**: puzzle(state(1,2,3,\*,8,5,4,7,6)).

**Hard**: puzzle(state(2,3,\*,1,8,5,4,7,6)).

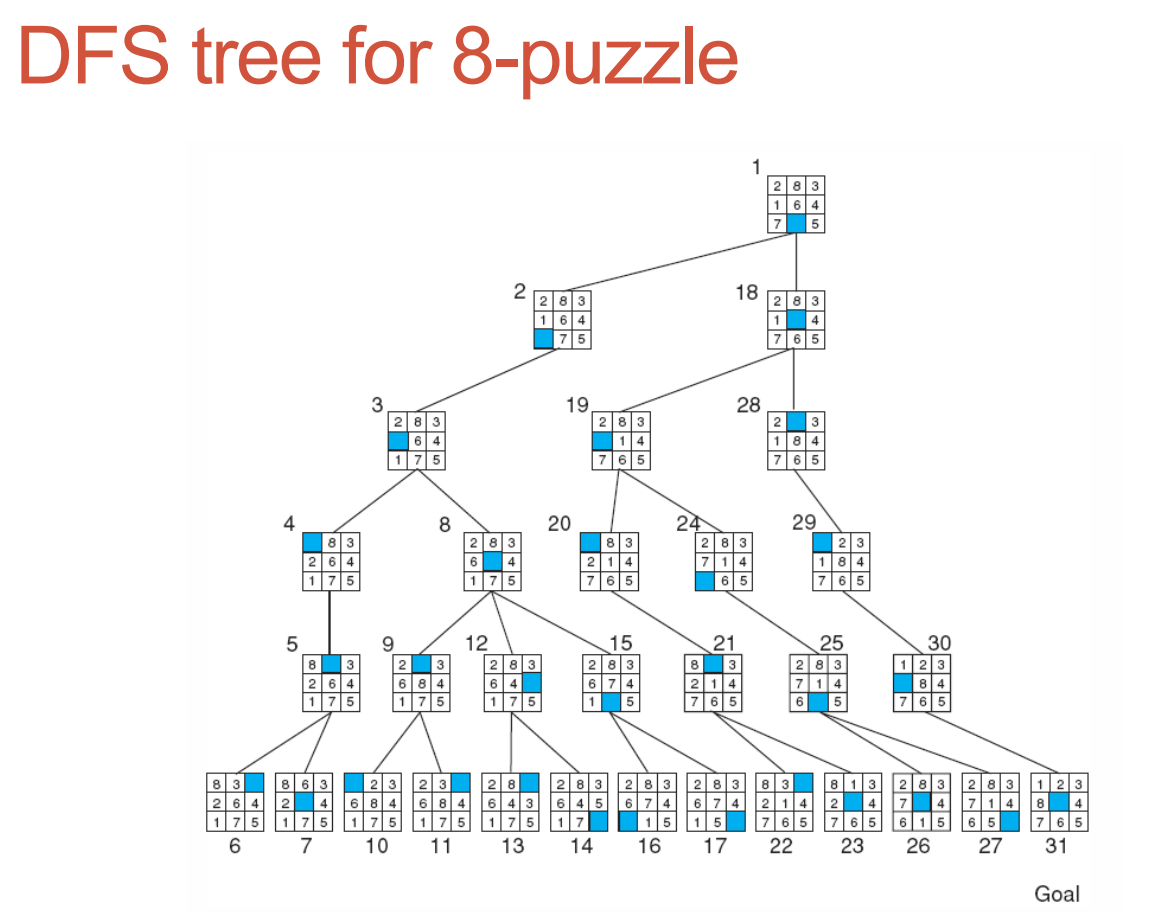
**Hardest** (Computable): puzzle(state(4,1,5,7,\*,2,8,3,6)).

**Hardest** (Not Computable): puzzle(state(2,3,5,1,\*,4,8,7,6)).

WORST CASE – (m >> b)

Time Complexity – O(bm) **Too High** and will take **Hours** to Solve it.



3.) Uniform Cost Search

Uniform-cost search expands the node **n** with the **lowest path cost**

* For each node n, keep track of the “**path cost**”, g(n)
* Maintain frontier as a **priority queue**

**Differences from BFS**:

* Must check for goal when node chosen for expansion (instead of when generated)
* Must also check for each state generated that is in frontier, whether this new path has lower path cost.

**Code**

*% Breadth For Search Implementation of the 8 Puzzle Problem*

*% There are Cases When the Stack Overflows (>1 GB), Due to Space Complexity of UFS O(b^d)*

*% Main F(x) to Solve the 8 Puzzle Problem*

puzzle(State)*:-*

    ucs([State],[State],N),*!*,

    write("Total Steps: "),

    write(N)*.*

*% Check if the Goal is Reached*

ucs([State | \_],\_,0)*:-*goal(State)*.*

ucs([CurState | RemQueue],Visited,N)*:-*

*% Find all the Possible Moves*

    findall(X,move(CurState,X,\_),AllPossibleState),

*% Avoid Repeated Nodes*

    removeDuplicate(AllPossibleState,Visited,PossibleState),

*% Store Frontier of Nodes in FIFO Queue*

    append(PossibleState,Visited,NewVisited),

*% Store Remaining in Other Queue*

    append(RemQueue,PossibleState,NewRemQueue),

*% Recur for Updated Queue with New Visited Nodes*

    ucs(NewRemQueue,NewVisited,N1),

*% Count the Required Number of Steps*

    N is N1 + 1*.*

*% F(x) to Remove Duplicates from the List*

*% Base Case*

removeDuplicate([],\_,[])*.*

*% If Member is Not in the List, then Add it to Answer*

removeDuplicate([H1|AllRem],Visited,[H1 | T1])*:-*

    not(memberchk(H1,Visited)),

    removeDuplicate(AllRem,Visited,T1)*.*

*% If Member is in the List, then Don't Add it to Answer*

removeDuplicate([H1|AllRem],Visited,T1)*:-*

    memberchk(H1,Visited),

    removeDuplicate(AllRem,Visited,T1)*.*

*% Goal State to be Acheived*

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

*% Reference for Below Moves - https://stackoverflow.com/questions/67642302/prolog-for-eight-puzzle*

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 )*.*

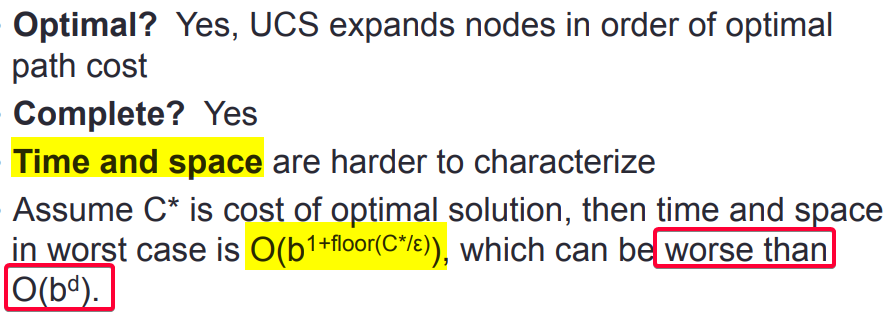
move( state(A,B,C,D,E,F,G,\*,J), state(A,B,C,D,\*,F,G,E,J), up   )*.*

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   )*.*

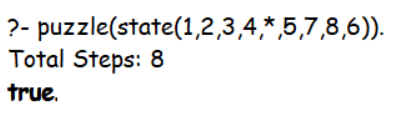
move( state(A,B,C,D,E,F,G,H,\*), state(A,B,C,D,E,F,G,\*,H), left )*.*

**Analysis**

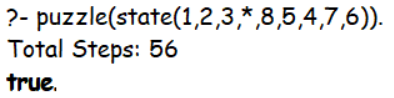


**Output**

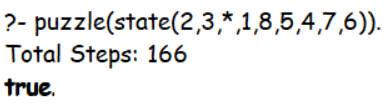
**Easy**: puzzle(state(1,2,3,4,\*,5,7,8,6)).



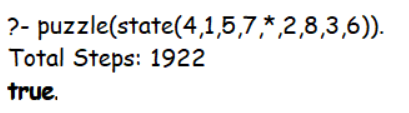
**Medium**: puzzle(state(1,2,3,\*,8,5,4,7,6)).



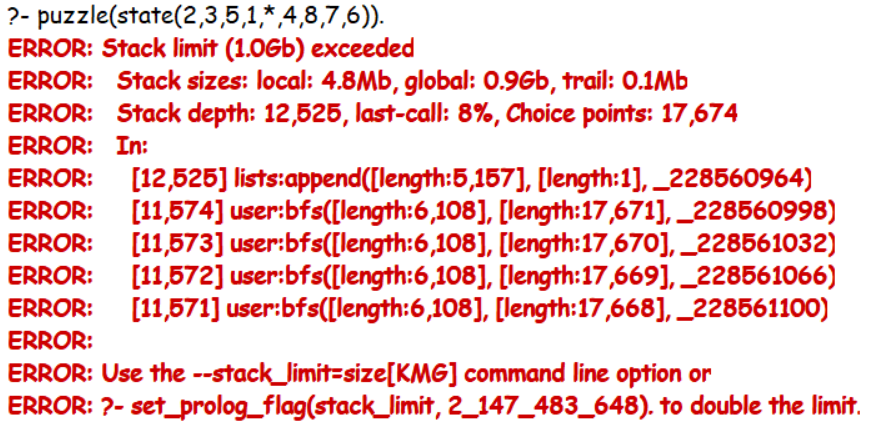
**Hard**: puzzle(state(2,3,\*,1,8,5,4,7,6)).



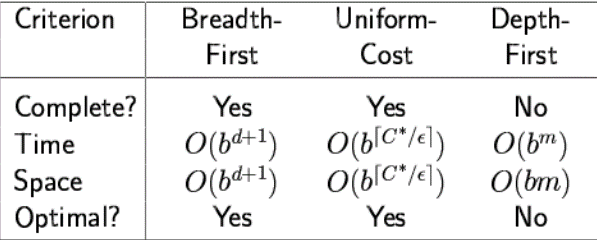
**Hardest** (Computable): puzzle(state(4,1,5,7,\*,2,8,3,6)).



**Hardest** (Not Computable): puzzle(state(2,3,5,1,\*,4,8,7,6)).



Q2.) Compare Algorithms.



Q3.) Which Algorithm is **Best** suited for implementing **8 Puzzle problem** and why?

For **Smaller Test Cases**, Depth for Search Works **Fast** and is **Efficient** with Space Complexity, But Only *Drawback* is that in case it Gets in Wrong Branch {in Tricky Cases} and Keep Going Deeper and Deeper, it Will Lead to **Infinite Path Problem.**

**Breadth for Search** is better in Case where we want the **Right Answer** for Smaller Depth, Even though we are using **Lots of** **Space**. It may or may **not** give the **Optimal Path** to Reach the Goal State.

**SUBMITTED BY**: U19CS012

BHAGYA VINOD RANA