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

Criteria (b is branching factor; d is depth of goal):

- Complete? Yes (if some goal at finite depth *d, and b is finite*)
- Space? Not great, size of frontier, so $O(b^d)$ potentially
- Time? Nodes generated, $b + b^2 + b^3 + \ldots + b^d = O(b^d)$
- Optimal? Yes, if all actions have same cost

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

Output

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

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

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

```
?- puzzle(state(1,2,3,*,8,5,4,7,6)).
Total Steps: 56
true.
```

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

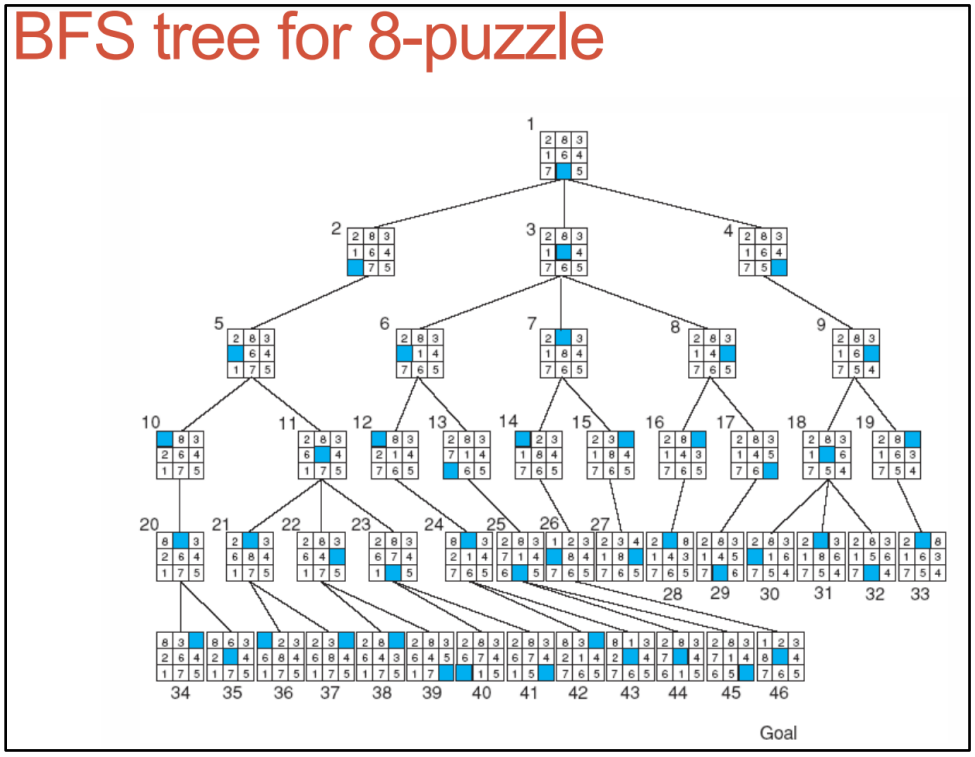
```
?- puzzle(state(2,3,*,1,8,5,4,7,6)).
Total Steps: 166
true.
```

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

```
?- puzzle(state(4,1,5,7,*,2,8,3,6)).
Total Steps: 1922
true.
```

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

```
?- puzzle(state(2,3,5,1,*,4,8,7,6)).
ERROR: Stack limit (1.0Gb) exceeded
ERROR:   Stack sizes: local: 4.8Mb, global: 0.9Gb, trail: 0.1Mb
ERROR:   Stack depth: 12,525, last-call: 8%, Choice points: 17,674
ERROR:   In:
ERROR:     [12,525] lists:append([length:5,157], [length:1], _228560964)
ERROR:     [11,574] user:bfs([length:6,108], [length:17,671], _228560998)
ERROR:     [11,573] user:bfs([length:6,108], [length:17,670], _228561032)
ERROR:     [11,572] user:bfs([length:6,108], [length:17,669], _228561066)
ERROR:     [11,571] user:bfs([length:6,108], [length:17,668], _228561100)
ERROR:
ERROR: Use the --stack_limit=size[KMG] command line option or
ERROR: ?- set_prolog_flag(stack_limit, 2_147_483_648). to double the limit.
```



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

Complete? No: fails in infinite-depth spaces with loops, but is complete in finite spaces (when avoiding repeated states)

Optimal? No.

Time? $O(b^m)$, where *m* is maximum depth of any node. Bad if *m* is much larger than *d*

Space (only good thing!): Need only store path from root of search tree and siblings of those nodes, so O(bm)

Output

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

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

start

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 = 2
true.
```

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

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

start

1 2 3
* 8 5
4 7 6

down

1 2 3
4 8 5
* 7 6

left

1 2 3
4 8 5
7 * 6

up

1 2 3
4 * 5
7 8 6

right

1 2 3
4 5 *

right

1 2 3
4 5 *
7 8 6

down

1 2 3
4 5 6
7 8 *

moves = 5
true.
```

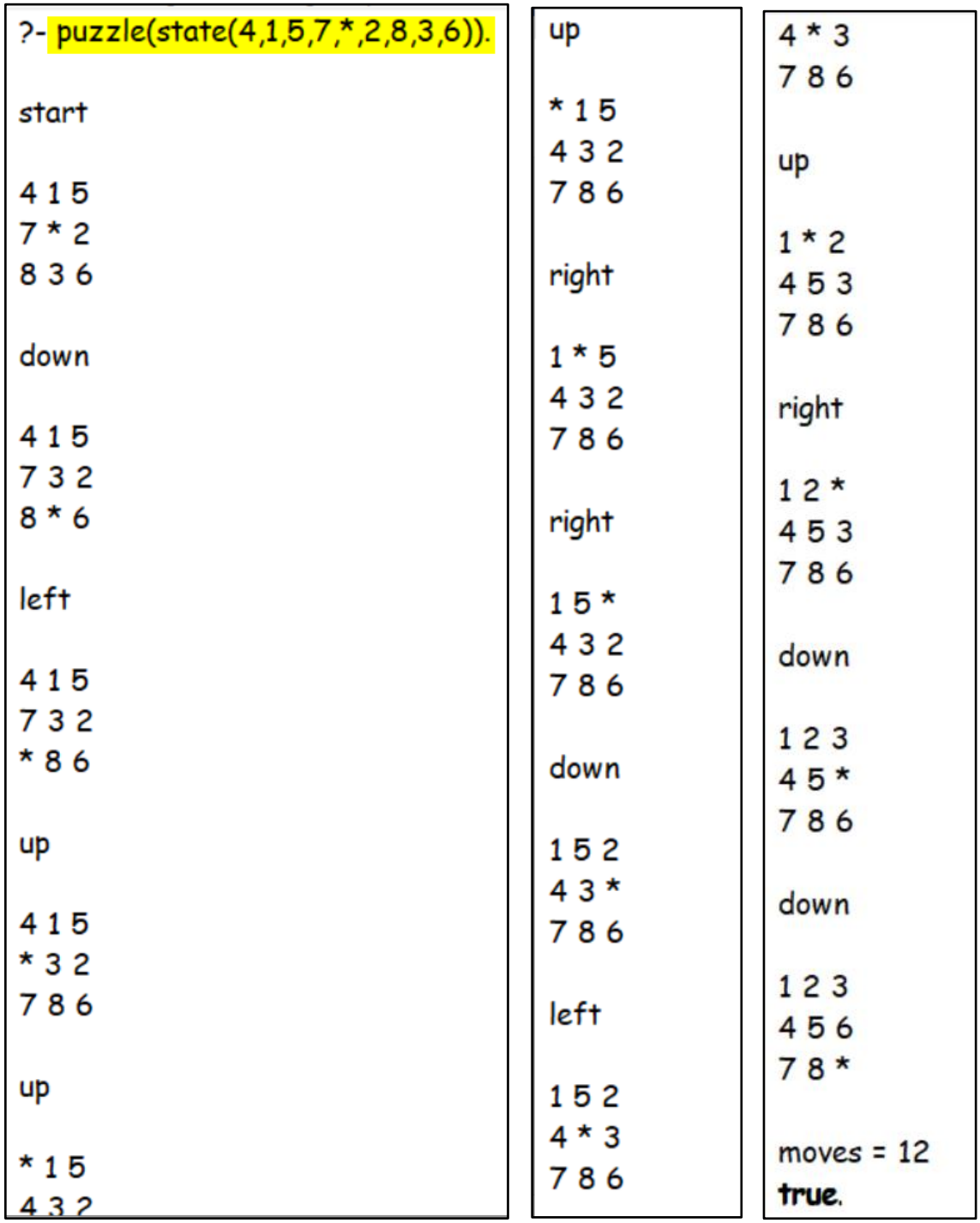
Hard: puzzle(state(2,3,*,1,8,5,4,7,6)).

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

start

2 3 *
1 8 5
4 7 6

left

2 * 3
1 8 5
4 7 6

left

* 2 3
1 8 5
4 7 6

down

1 2 3
* 8 5
4 7 6

down

1 2 3
4 8 5
```

```
down

1 2 3
* 8 5
4 7 6

down

1 2 3
4 8 5
* 7 6

left

1 2 3
4 8 5
7 * 6

up

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 = 8
true.
```

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

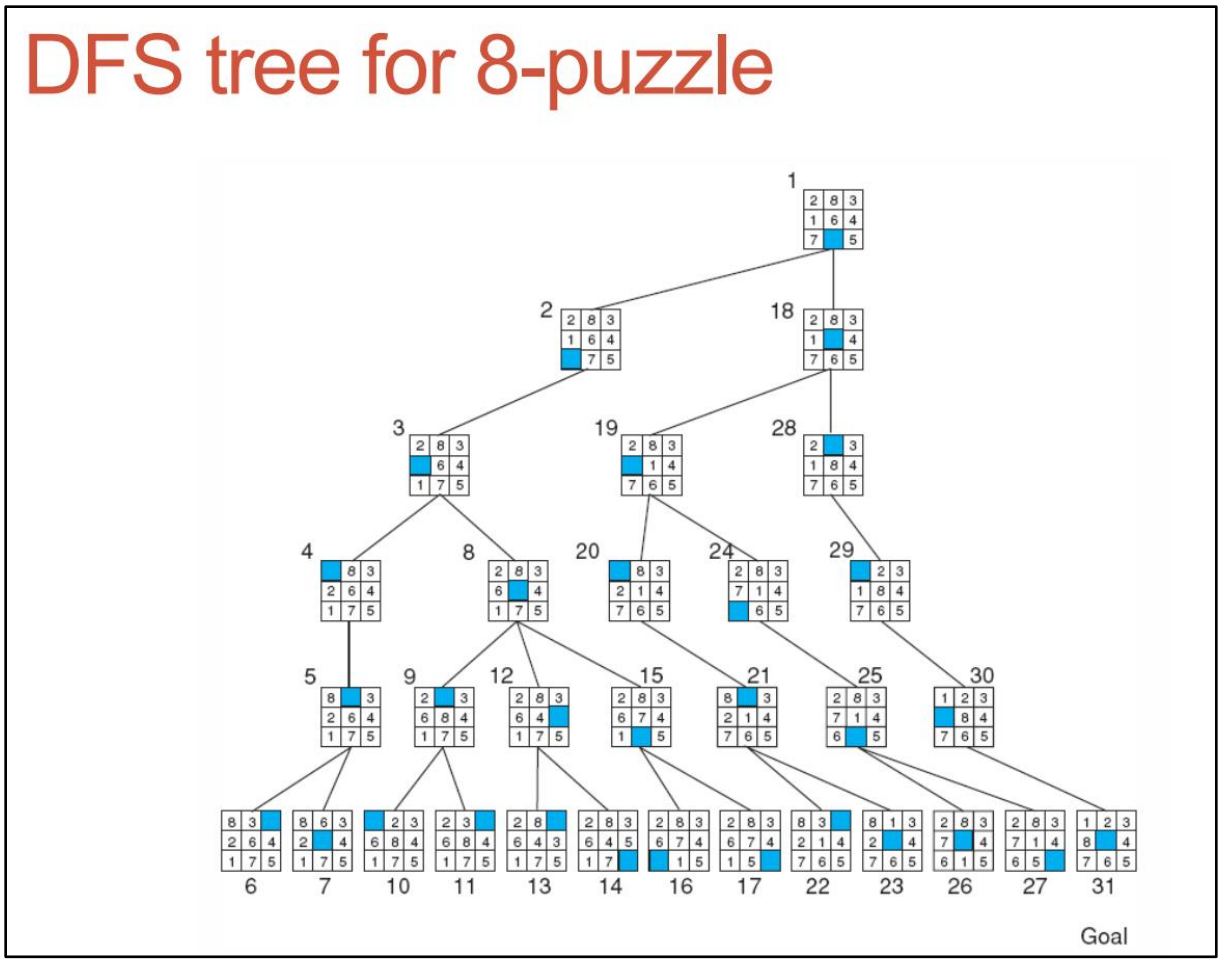


```
?- puzzle(state(4,1,5,7,*,2,8,3,6)).

start

4 1 5
7 * 2
8 3 6

down

4 1 5
7 3 2
8 * 6

left

4 1 5
7 3 2
* 8 6

up

4 1 5
* 3 2
7 8 6

up

* 1 5
4 3 2
7 8 6

right

1 * 5
4 3 2
7 8 6

right

1 5 *
4 3 2
7 8 6

down

1 5 2
4 3 *
7 8 6

left

1 5 2
4 * 3
7 8 6

4 * 3
7 8 6

up

1 * 2
4 5 3
7 8 6

right

1 2 *
4 5 3
7 8 6

down

1 2 3
4 5 *
7 8 6

down

1 2 3
4 5 6
7 8 *

moves = 12
true.
```

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

WORST CASE – (m >> b)

Time Complexity – $O(b^m)$ **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

Optimal? Yes, UCS expands nodes in order of optimal path cost

Complete? Yes

Time and space are harder to characterize

Assume C* is cost of optimal solution, then time and space in worst case is $O(b^{1+\text{floor}(C^*/\varepsilon)})$, which can be worse than $O(b^d)$.

Output

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

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

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

```
?- puzzle(state(1,2,3,*,8,5,4,7,6)).
Total Steps: 56
true.
```

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

```
?- puzzle(state(2,3,*,1,8,5,4,7,6)).
Total Steps: 166
true.
```

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

```
?- puzzle(state(4,1,5,7,*,2,8,3,6)).
Total Steps: 1922
true.
```

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

```
?- puzzle(state(2,3,5,1,*,4,8,7,6)).
ERROR: Stack limit (1.0Gb) exceeded
ERROR:   Stack sizes: local: 4.8Mb, global: 0.9Gb, trail: 0.1Mb
ERROR:   Stack depth: 12,525, last-call: 8%, Choice points: 17,674
ERROR:   In:
ERROR:     [12,525] lists:append([length:5,157], [length:1], _228560964)
ERROR:     [11,574] user:bfs([length:6,108], [length:17,671], _228560998)
ERROR:     [11,573] user:bfs([length:6,108], [length:17,670], _228561032)
ERROR:     [11,572] user:bfs([length:6,108], [length:17,669], _228561066)
ERROR:     [11,571] user:bfs([length:6,108], [length:17,668], _228561100)
ERROR:
ERROR: Use the --stack_limit=size[KMG] command line option or
ERROR: ?- set_prolog_flag(stack_limit, 2_147_483_648). to double the limit.
```

Q2.) Compare Algorithms.

Criterion	Breadth-First	Uniform-Cost	Depth-First
Complete?	Yes	Yes	No
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$
Optimal?	Yes	Yes	No

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