Eight Puzzle

The puzzle consists of eight sliding tiles, numbered by digits from 1 to 8, and arranged in a 3 by 3 array of nine cells. One of the cells is always empty, and any adjacent tile can be moved into the empty cell. We can say that the empty cell is allowed to move around, swapping its place with any of the adjacent tiles.

The initial state is some arbitrary arrangement of the tiles. An initial state and the goal state are shown next.

an initial state

1	2	3
8		5
7	4	6

goal state

1	2	3
8		4
7	6	5

For the initial state above, there are four possible moves. Tile 2, 8, 5 or 4 can be moved into the empty cell, e.g. one state transitions is shown by:

current state

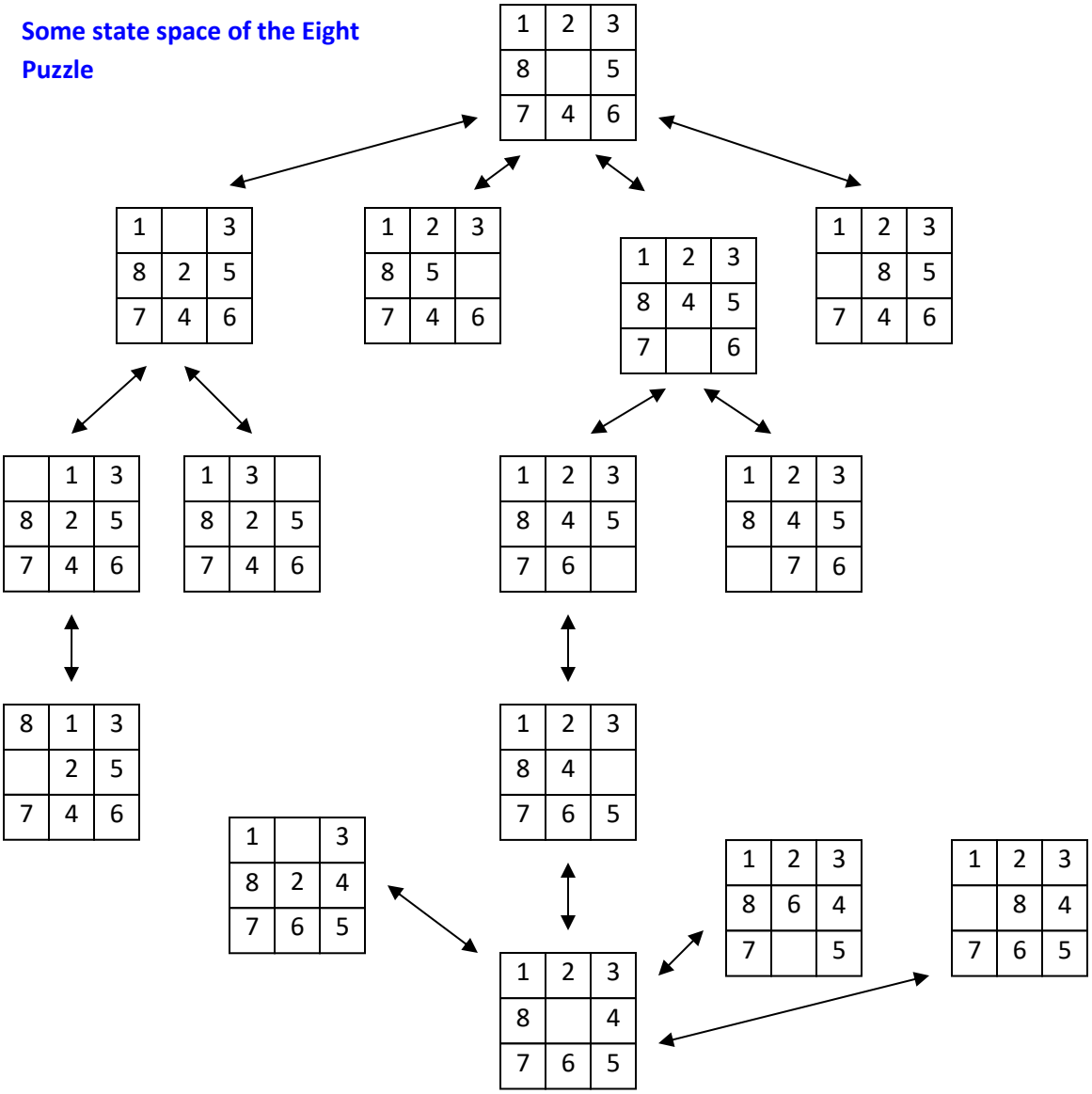
1	2	3
8		5
7	4	6

a successor or child state

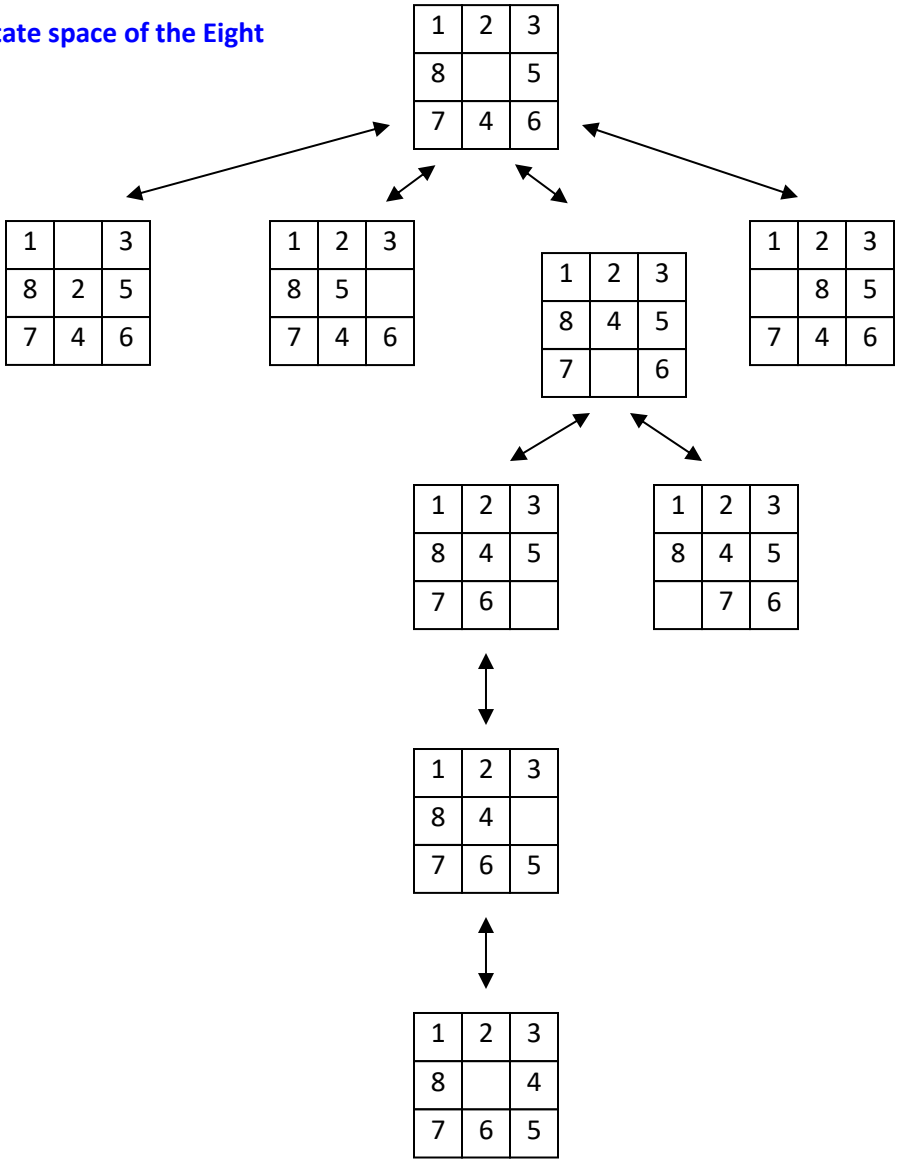
1	2	3
8	4	5
7		6

A move or state transition is only allowed if the ***Manhattan*** distance between the current and new states is 1. The problem is to find a sequence of moves from the initial state to the goal state.

Some state space of the Eight Puzzle



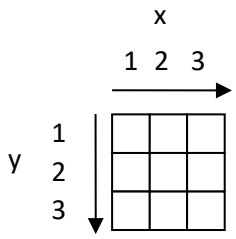
Some state space of the Eight Puzzle



Representing and Solving the 8-Puzzle in Prolog

Note that most of the Prolog program code below is outside the scope of this course. It is only included to show the power of Prolog for this type of programming and to show how a state might be represented, and of course for your interest.

First we need a way to represent and store a state. We can think of the tiles as being numbered from 1 to 8. One possibility is to assign each tile an (x, y) pair of co-ordinates. We can think of the empty space as tile0.



So in the following state

1	5	3
8		2
7	4	6

tile1 is at (1, 1), tile2 at (3, 2), tile3 at (3, 1). Empty position or tile0 if you like is at (2,2).

We wish to store all the tile positions together and a natural way to do this is put them in a list like

[t0, t1, t2, t3, t4, t5, t6, t7, t8].

In the above state t0 = (2,2), t1 = (1,1) etc. The whole state could be written as:

[(2,2), (1,1), (3,2), (3,1), (2,3), (2,1), (3,3), (1,3), (1,2)]

Similarly, the goal state

1	2	3
8		6
7	5	4

could be represented by

[(2,2), (1,1), (2,1), (3,1), (3,3), (2,3), (3,2), (1,3), (1,2)].

We can use the following Prolog code to represent the goal state with:

```
goal([(2,2),(1,1),(2,1),(3,1),(3,2),(3,3),(2,3),(1,3),(1,2)]) .
```

Generating Successor States

First of all the predicate

```
%  move( State1, State2 )   generates a successor state  %
```

is required. One way to implement this is:

```
move( [E | Tiles] , [T| Tiles1] ):-
      swap( E , T , Tiles , Tiles1 ) .

swap( E , T , [T | Ts] , [E | Ts] ):-
      mandist( E , T , 1 ) .

swap( E , T , [T1 | Ts] , [T1 | Ts1] ):-
      swap( E , T , Ts , Ts1 ).
```

The predicate move/2 passes the job onto an auxiliary predicate swap/4. It asks swap to choose a tile position T from the list Tiles, then replace it with E and call the updated list Tiles1.

swap does this by first checking to see if tile1 whose position is T can be moved which is possible if the Manhattan distance between T and E is 1. If that fails, swap recursively iterates thru the rest of the list Ts and it will pick a tile position T to swap with E if the distance equals 1. A successor state can be constructed by removing T from Ts, replacing it with E, calling the updated tiles list Ts1.

The Manhattan distance can be computed as follows:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Manhattan Distance - mandist(TilePos1 ,TilePos2,Dist)  %
%   is the distance between two tile positions .                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

mandist( (X,Y) , (X1,Y1) , D ):-
      diff( X , X1 , Dx ) ,
      diff( Y , Y1 , Dy ) ,
      D is Dx + Dy .

diff( A , B , D ):-
    D is A - B , D > 0 , !
      ;
      D is B - A .
```

More Prolog Code

The code below can help you get the program running and choose a starting state.

```
dfs :-      write('Start at depth 4 5 6 7 8 or 18 ?  : '),
      read( I ),
      start( I , X ),
      solve( X, [], Sol ),

    reverse(Sol, Soln),
      nl, showSolPath( Soln).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     The goal node and some starting nodes                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

goal( [(2,2), (1,1), (2,1), (3,1), (3,2), (3,3), (2,3), (1,3), (1,2)]).

%depth 4
start4( [(2,2), (1,1), (3,2), (2,1), (3,1), (3,3), (2,3), (1,3), (1,2)]).

%depth 5
start5( [(2,3), (1,2), (1,1), (3,1), (3,2), (3,3), (2,2), (1,3), (2,1)]).

%depth 6
start6( [(1,3), (1,2), (1,1), (3,1), (3,2), (3,3), (2,2), (2,3), (2,1)]).

%depth 7
start7( [(1,2), (1,3), (1,1), (3,1), (3,2), (3,3), (2,2), (2,3), (2,1)]).

%depth 8
start8( [(2,2), (1,3), (1,1), (3,1), (3,2), (3,3), (1,2), (2,3), (2,1)]).

%depth 18
start18( [(2,2), (2,1), (1,1), (3,3), (1,2), (2,3), (3,1), (1,3), (3,2)]).


% predicate to help you choose a starting state with
% solution at different depths
start( I , X ) :-
        I == 4 , start4( X ) , !
      ;
        I == 5 , start5( X ) , !
      ;
        I == 6 , start6( X ) , !
      ;
        I == 7 , start7( X ) , !
      ;
        I == 8 , start8( X ) , !
      ;
        I == 18, start18( X ) .
```

Exercise
They following starting states require 4 steps, 5 steps and 18 steps respectively to the goal state. Represent the them in Prolog and run the eight-puzzle program on them.

Check the number of steps in the solution of each. What happens with the last one?

1	3	4
8		2
7	6	5

2	8	3
1	6	4
7		5

2	1	6
4		8
7	5	3

Alternative Representation

Coming in 2017.