

Laboratory - 3

implement A* search algorithm
8 puzzle problem.

initial

5	4	
6	1	8
7	3	2

final

1	2	
3	4	5
6	7	8

i) Heuristic: Number tiles out of place.

5	4	
6	1	8
7	3	2

$$h = 1 + 1 + 1 + 1 + 1 + 1 = 8$$

$$g = 0$$

$$h = 8$$

5	4	
6	1	8
7	3	2

$$h = 8$$

$$g = 1$$

$$h = 8$$

$$g = 1$$

$$h = 8$$

$$g = 2$$

5	4	
6	1	8
7	3	2

$$h = 8$$

$$g = 2$$

5	4	
6	1	8
7	3	2

$$h = 8$$

$$g = 2$$

$$h = 8$$

$$g = 2$$

5	4	
6	1	8
7	3	2

$$h = 8$$

$$g = 2$$

$$h = 8$$

$$g = 3$$

5	4	
6	1	8
7	3	2

$$h = 8$$

$$g = 3$$

$$h = 7$$

$$g = 3$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 3$$

$$h = 7$$

$$g = 4$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 4$$

$$h = 7$$

$$g = 4$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 4$$

$$h = 7$$

$$g = 5$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 5$$

$$h = 7$$

$$g = 5$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 5$$

$$h = 7$$

$$g = 6$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 6$$

$$h = 7$$

$$g = 6$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 6$$

$$h = 7$$

$$g = 6$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 6$$

$$h = 7$$

$$g = 6$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 6$$

$$h = 7$$

$$g = 6$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 6$$

$$h = 7$$

$$g = 6$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 6$$

$$h = 7$$

$$g = 6$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 6$$

$$h = 7$$

$$g = 6$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 6$$

$$h = 7$$

$$g = 6$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 6$$

$$h = 7$$

$$g = 6$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 6$$

$$h = 7$$

$$g = 6$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 6$$

$$h = 7$$

$$g = 6$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 6$$

$$h = 7$$

$$g = 6$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 6$$

$$h = 7$$

$$g = 6$$

5	4	
6	1	8
7	3	2

$$h = 7$$

$$g = 6$$

<math display="

s	i	g
3	.	8
G	7	2

g_{26}
 h_{24}

g_{27}	s	i	g	g_{27}	s	i	g	g_{27}	s	i	g
$h=5$	3	2	8	$h=4$	3	8	-	$h=5$	3	7	8
	6	7	2		6	7	2		6	7	2

~~\$1~~

g_{28}	s	i	1	g	s	i	4	g_{28}
$h=4$	3	8	4		3	8	2	$h=8$
	6	7	2		6	7		

g_{29}	s	i	1	g	s	i	4	g_{29}
$h=5$	3	8	4		3	8	2	$h=5$
	6	7	2		6	7	2	

algorithm

```
function N-STAR misplace-Tile(misplaced-tilesstart-state, goal-state)
    open-list ← priority queue containing start state
    with f(start-state)=0.

    closed-list ← empty set;
    g[start-state] ← 0.
    parents[start-state] ← None.

    while open-list is not empty:
        current-state ← state in open-list with lowest
        if current state equals goal state:  $f(n)$ 
            return reconstruct path(parents, current-state).
        remove current-state from open-list
        add current-state to closed-list
        for each neighbor of current state:
            if neighbor is in closed-list:
                continue.
            tentative-g ← g[current-state]+1
            if neighbor is not in open-list or
            tentative-g < g[neighbor]:
                parents[neighbor] ← current state
                g[neighbor] ← tentative-g
                f[neighbor] ← g[neighbor]+misplaced-tiles(misplaced-tilesneighbor, goal-state)

            if neighbor is not in open-list:
                add neighbor to open-list with priority
                f[neighbor]

    return failure.
```

```
function misplaced-tiles(state, goal-state)
    counts ← 0
```

for each tile in state:

if tile is ^{not} in ~~in~~ its correct position
and tile is not blank:

 count += 1

return count.

function reconstruct_path(parents, current_state)

path = empty list

while current state is not None:

 add current state to path.

 current state = parents[current_state]

return reverse(path)

Jan
15/02/20

A* algorithm with Manhattan distance

function A-star(start, goal, grid)

 openset = priorityQueue()

 closedset = set()

 start.g = 0

 start.h = manhattan_dist(start, goal)

 start.f = start.g + start.h

 openset.push(start)

 while openset is not empty:

 current = openSet.pop()

 if current == goal:

 return reconstruct_path(current)

 closedSet.add(current)

 for neighbour in current.neighbors:

 if neighbour in closedSet:

 continue

 tentative_g = current.g + dist_bw(current, neighbour)

 if neighbour not in open set:

 openSet.push(neighbour)

 if tentative_g >= neighbour.g:

 continue

 neighbour.cameFrom = current

 neighbour.g = current.tentative_g

 neighbour.h = manhattan_dist(neighbour, goal)

 neighbour.f = neighbour.g + neighbour.h

 return None

function manhattan_dist(node, goal)

return abs(node.x - goal.x) + abs(node.y - goal.y)

function reconstruct_path(node)

total_path = [node]

while node.comefrom is not None:

node = node.comefrom

total_path.prepend(node)

return total_path.

State space tree.

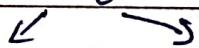
$g=0$	2	8	3
$h=6$	2	6	4
	7	5	

2	2	3
8		4
7	6	5

initial state

Goal state

$$h(n) = 6$$



$h=7$

$g=1$

2	8	3
6	4	
1	7	5

2	8	3
1	6	4
7	5	

$h=5$

$g=1$

$h=4$

$g=2$

2	8	3
1	7	4
7	6	5

2	8	3
1	6	4
7	5	

2	8	3
1	6	4
7	5	

$h=6$

$g=2$

$h=5$

$g=3$

2	8	3
1	7	4
7	6	5

2	8	3
1	7	4
7	6	5

2	8	3
1	7	4
7	6	5

2	8	3
1	7	4
7	6	5

2	8	3
1	7	4
7	6	5

$h=2$

$g=4$

2	3
1	8
7	6

2	3
1	8
7	6

$h=3$

$g=3$

1	2	3
8	4	
7	6	5

$h=1$

$g=5$

1	2	3
8	9	
7	6	5

 $h=1$ $j=5$

$h=0$	1	2	3
$g=6$	8	0	9
	7	6	5

1	2	3
7	8	9
6	5	

 $h=2$ $j=6$ Croat