PROJECT REPORT

8 PUZZLE PROBLEM

USING

GREEDY BEST FIRST SEARCH

भारतीय सूचना प्रौद्योगिकी संस्थान भागलपुर
Indian Institute of Information Technology
Bhagalpur

Department of Computer Science and Engineering

Indian Institute of Information

Technology, Bhagalpur – 813210

Analysis and Implementation of Puzzle Problem:

Project submitted in
Sept 2019
to the department of
Computer Science and Engineering of

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY
BHAGALPUR

in partial fulfilment of the requirements
for the degree of
Bachelor of Technology
in
Computer Science and Engineering
By

ABHISHEK KUMAR 170101003 CSE
HIMANSHU RANJAN 170101017 CSE
HITESH KUMAR SAHU 170102019 CSE
SURAJ KUMAR 170101052 CSE
SANTOSH KUMAR 170102043 ECE

Under the humble guidance of

Rupam Bhattacharyya

8 Puzzle (The Problem)

The 8-puzzle is a **sliding tile puzzle** that is made up of a square structured frame area containing 9 tiles in random/irregular order with one tile missing. The 8-puzzle problem is a puzzle invented and popularized by **Noyes Palmer Chapman** in the 1870s.

It is a smaller version of the **15-puzzle** (also called Gem Puzzle, Boss Puzzle, Game of Fifteen, Mystic Square and numerous other names). 8-puzzle is basically a frame area separated into 3x3 grids containing 8 tiles and one void grid. The tiles are marked in some way so as they can be identified. The tiles are mostly numbered from 1 to 8. We are given with an initial configuration of the tiles. A desired final configuration is also given. We have to reach the final state by sliding the tiles in either vertically or horizontally using the empty grid present. Diagonally swapping is not allowed.

Problem Approach:

The complexity of possible moves toward the final solution in a game like this is great. It can take an average computer great lengths of time to find the correct sequences for a particular configuration of the 8 puzzle game if the search method is "blind". The problem is how do you make the computer order the moves intelligently to find the shortest path to the winning game state? Solving a problem such as this can be done two ways:

1. ***Guess through every possible state by doing a blind search of the state space.***

2. ***Implement a search operation that is guaranteed to find the shortest solution using "Informed methods" (How?).***

Informed Search Strategies:

Informed methods help us gain information about solving a problem through its current state space. This keeps a program from blundering around blindly guessing. Informed search strategies make use of information about the path of moves we took to get where we are currently by using an "evaluation function".

Greedy best-first search:

Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function; that is, $f(n)=h(n)$.

f(n) = Estimate of cost from n to goal

Greedy best-first search expands the node that appears to be closest to goal.

1. If the successor's heuristic is better than its parent, the successor is set at the front of the queue (with the parent reinserted directly behind it), and the loop restarts.

2. Else, the successor is inserted into the queue (in a location determined by its heuristic value). The procedure will evaluate the remaining successors (if any) of the parent.

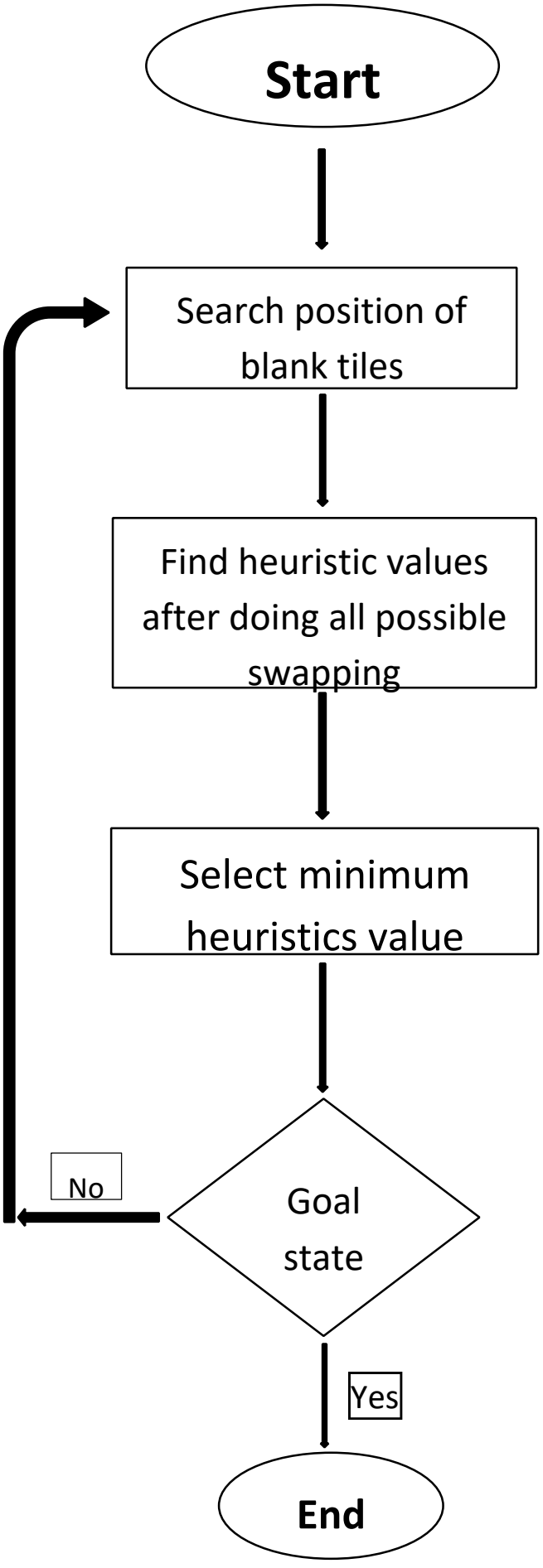
Properties of greedy best-first search

Complete: No – can get stuck in loops.

Time: O(b^m) - But a good heuristic can give dramatic improvement
Space: O(b^m) - keeps all nodes in memory

Optimal: No

FLOW CHART:


Start
Search position of blank tiles
Find heuristic values after doing all possible swapping
Select minimum heuristics value
Goal state
No
Yes
End

Source Code:

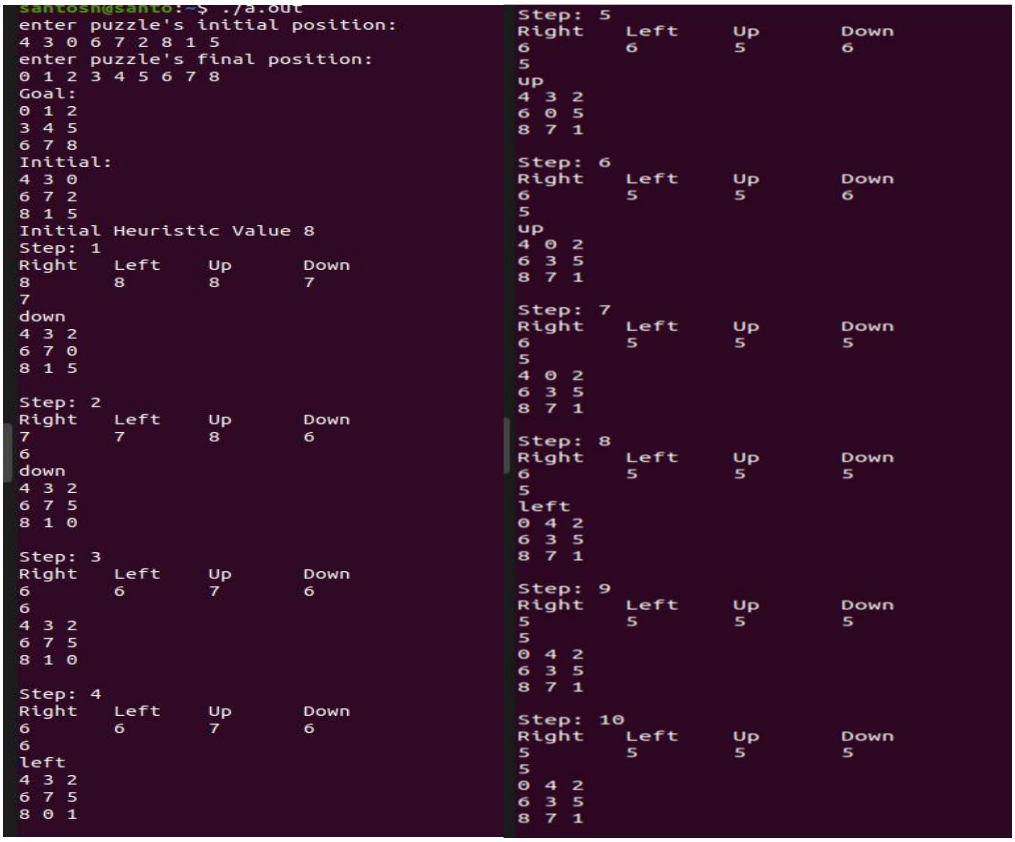
```
#include<stdio.h>
#include<stdlib.h>
int difference(int arr_in[][3], int arr_fi[][3]) //heuristic funtion
{
	int counter =0 ,i,j;
for(i=0;i<3; i++)
for( j=0;j<3;j++)
	{
	if(arr_in[i][j]!=0)
	{
	if(arr_in[i][j] != arr_fi[i][j])
	counter++;
	}
	}
	return counter;
}
void display(int arr_in[][3]) //to display the puzzle
{
	int i,j;
for(i=0;i<3;i++)
	{
	for(j=0;j<3;j++)
	printf("%d ", arr_in[i][j]);
printf("\n");
	}
}
int diffup(int arr_in[][3], int arr_fi[][3]) //calculate heruristic value if up operation is done
{
	int temp[3][3], i, j;
for(i=0;i<3;i++)   for
(j=0;j<3;j++)
	temp[i][j] = arr_in[i][j];
for(i=1;i<3;i++)   for
(j=0;j<3;j++)
	if(arr_in[i][j]==0)
	{
		temp[i-1][j] = 0;
		temp[i][j] = arr_in[i-1][j];
	}
	return difference(temp, arr_fi);
}
int diffdown(int arr_in[][3], int arr_fi[][3]) //calculate heruristic value if down operation is done
{
	int temp[3][3], i, j;
for(i=0;i<3;i++)   for
(j=0;j<3;j++)
	temp[i][j] = arr_in[i][j];
for(i=0;i<2;i++)   for
(j=0;j<3;j++)
	if(arr_in[i][j]==0)
	{
		temp[i+1][j] = 0;
		temp[i][j] = arr_in[i+1][j];
```

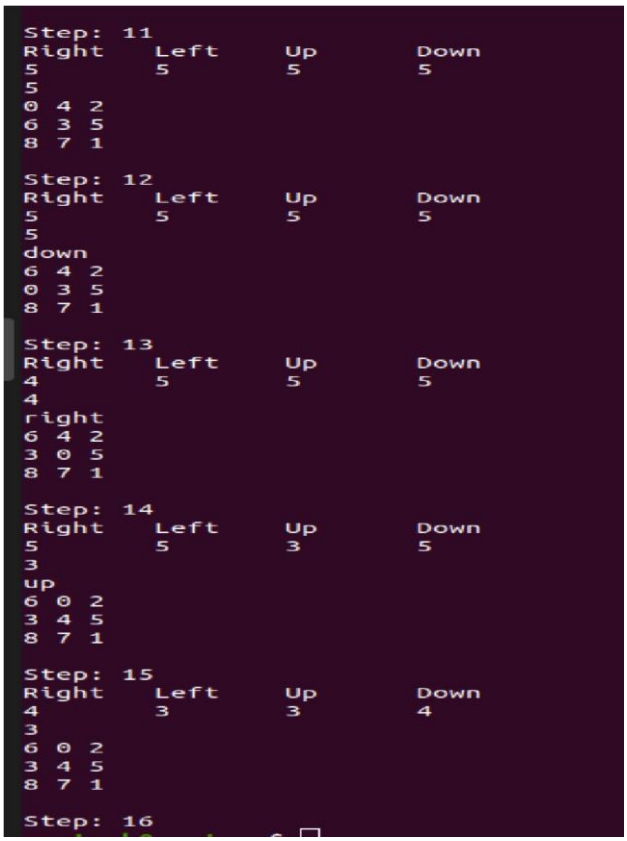
```
		}
		return difference(temp, arr_fi);
}
int diffleft(int arr_in[][3], int arr_fi[][3]) //calculate heruristic value if left operation is done
{
		int temp[3][3], i, j;
for(i=0;i<3;i++)	for
(j=0;j<3;j++)		temp[i][j] =
arr_in[i][j];
for(i=0;i<3;i++)	for
(j=1;j<3;j++)
		if(arr_in[i][j]==0)
		{
			temp[i][j-1] = 0;
			temp[i][j] = arr_in[i][j-1];
		}
		return difference(temp, arr_fi);
}
int diffright(int arr_in[][3], int arr_fi[][3]) //calculate heruristic value if right operation is done
{
		int temp[3][3], i, j;
for(i=0;i<3;i++)	for
(j=0;j<3;j++)		temp[i][j] =
arr_in[i][j];
for(i=0;i<3;i++)	for
(j=0;j<2;j++)
		if(arr_in[i][j]==0)
		{
			temp[i][j+1] = 0;
			temp[i][j] = arr_in[i][j+1];
		}
		return difference(temp, arr_fi);
}
int minimum (int a, int b, int c , int d) //returns minimum value among all possible heuristic value
{
		int min = a;
if(b<min)
min= b;
if(c<min)		min
= c;		if(d<min)
min = d;
		return min;
}
int puzzle(int arr_in[][3], int arr_fi[][3]) //solves puzzle
{
		int dup, ddown, dleft, dright;
		int temp, i , j,serial=0;
char ran[4];
		dup = diffup(arr_in,arr_fi);
		ddown = diffdown(arr_in,arr_fi); dleft =
		diffleft(arr_in,arr_fi); dright = diffright(arr_in,arr_fi);
		printf("Right\tLeft\tUp\tDown\n");
		printf("%d\t%d\t%d\t%d\n",  dright, dleft, dup, ddown);
		int min = minimum(dup, ddown, dleft, dright);
		printf("%d\n", min);
if (min == dright)
```

```
ran[serial++] = 'r';          if
(min == dleft)
ran[serial++] = 'l';          if
(min == dup)
ran[serial++] = 'u';          if
(min == ddown)
ran[serial++] = 'd';
 int sel = rand()%serial; //selects operation to be done
            char change = ran[sel];
if(change == 'r')
            {
            for(i=0;i<3;i++)
for (j=0;j<2;j++)
if(arr_in[i][j]==0)
            {
            arr_in[i][j] = arr_in[i][j+1];
arr_in[i][j+1] = 0;
printf("right\n");              return 0;
            }
            }
            else if(change == 'l')
            {
            for(i=0;i<3;i++)
for (j=1;j<3;j++)
if(arr_in[i][j]==0)
            {
            arr_in[i][j] = arr_in[i][j-1];
arr_in[i][j-1] = 0;
printf("left\n");    return 0;
            }
            }
            else if(change == 'u')
            {
            for(i=1;i<3;i++)
for (j=0;j<3;j++)
if(arr_in[i][j]==0)
            {
 arr_in[i][j] = arr_in[i-1][j];  arr_in[i-1][j] = 0;
            printf("up\n");
            return 0;
            }
            }
            else if(change == 'd')
            {
            for(i=0;i<2;i++)
            for (j=0;j<3;j++)
if(arr_in[i][j]==0)
            {
            arr_in[i][j] = arr_in[i+1][j];
arr_in[i+1][j] = 0;
printf("down\n");             return 0;
            }
            }
            return 0;
}
int main()
```

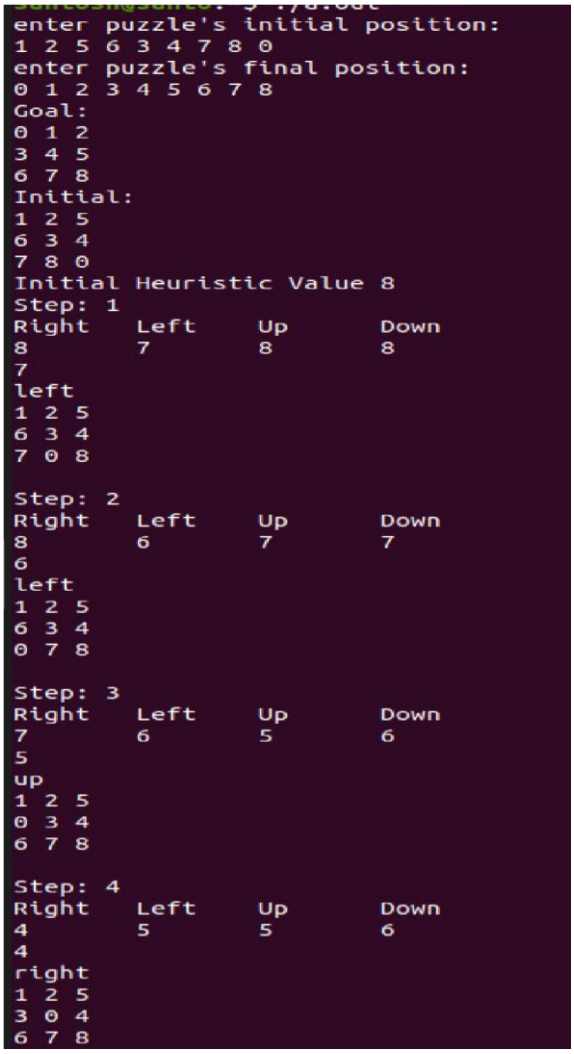
```
{
        int arr_in[3][3],arr_fi[3][3],i,j,d,steps=0;
        printf("enter puzzle's initial position: \n");
for(i=0;i<3;i++)   for(j=0;j<3;j++)
        {
        scanf("%d",&arr_in[i][j]);
        }
        printf("enter puzzle's final position: \n");
        for(i=0;i<3;i++)
for(j=0;j<3;j++)
        {
        scanf("%d",&arr_fi[i][j]);
        }
        printf("Goal: \n");
display(arr_fi);    printf("Initial: \n");
display(arr_in);  d = difference(arr_in, arr_fi);
printf("Initial Heuristic Value %d",d);
        while(1) //runs until goal state is not reached
        {
        d = difference(arr_in, arr_fi);
if(d==0)
        {
        printf("Goal state is reached in %d steps.", steps);
return 0;
        }
        steps++;
        printf("\nStep: %d \n", steps);
if(steps==16)     return 0;
        puzzle(arr_in, arr_fi);
        display(arr_in);
        }
        return 0;
}
```

Output 1:


santosh@santo:~$ ./a.out
enter puzzle's initial position:
4 3 0 6 7 2 8 1 5
enter puzzle's final position:
0 1 2 3 4 5 6 7 8
Goal:
0 1 2
3 4 5
6 7 8
Initial:
4 3 0
6 7 2
8 1 5
Initial Heuristic Value 8
Step: 1
Right Left Up Down
8 8 8 7
7
down
4 3 2
6 7 0
8 1 5
Step: 2
Right Left Up Down
7 7 8 6
6
down
4 3 2
6 7 5
8 1 0
Step: 3
Right Left Up Down
6 6 7 6
6
4 3 2
6 7 5
8 1 0
Step: 4
Right Left Up Down
6 6 7 6
6
left
4 3 2
6 7 5
8 0 1
Step: 5
Right Left Up Down
6 6 5 6
5
up
4 3 2
6 0 5
8 7 1
Step: 6
Right Left Up Down
6 5 5 6
5
up
4 0 2
6 3 5
8 7 1
Step: 7
Right Left Up Down
6 5 5 5
5
4 0 2
6 3 5
8 7 1
Step: 8
Right Left Up Down
6 5 5 5
5
left
0 4 2
6 3 5
8 7 1
Step: 9
Right Left Up Down
5 5 5 5
5
0 4 2
6 3 5
8 7 1
Step: 10
Right Left Up Down
5 5 5 5
5
0 4 2
6 3 5
8 7 1


Step: 11
Right Left Up Down
5 5 5 5
5
0 4 2
6 3 5
8 7 1
Step: 12
Right Left Up Down
5 5 5 5
5
down
6 4 2
0 3 5
8 7 1
Step: 13
Right Left Up Down
4 5 5 5
4
right
6 4 2
3 0 5
8 7 1
Step: 14
Right Left Up Down
5 5 3 5
3
up
6 0 2
3 4 5
8 7 1
Step: 15
Right Left Up Down
4 3 3 4
3
6 0 2
3 4 5
8 7 1
Step: 16

Output 2:

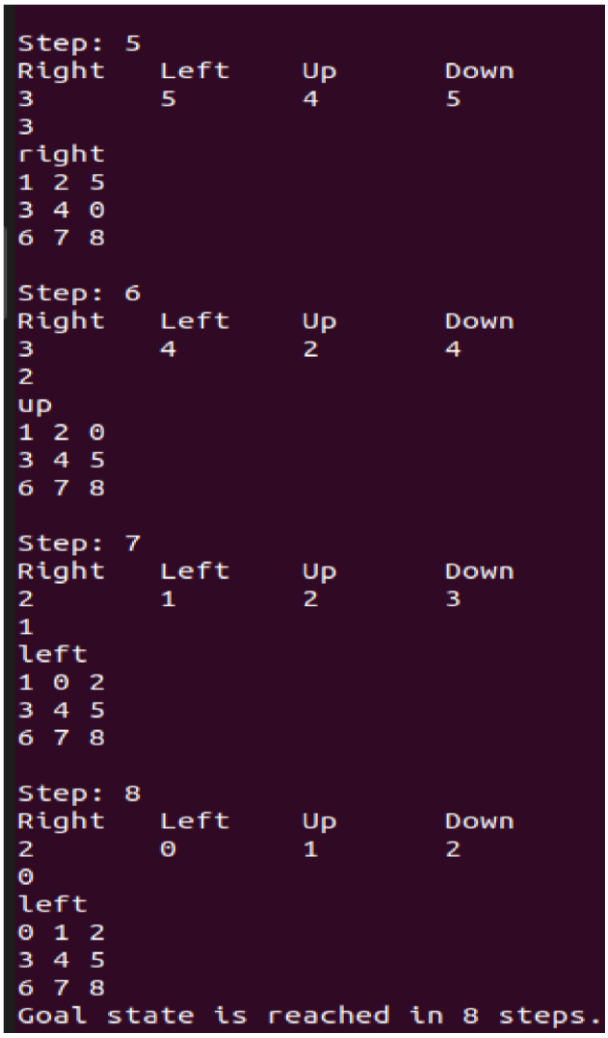


```
enter puzzle's initial position:
1 2 5 6 3 4 7 8 0
enter puzzle's final position:
0 1 2 3 4 5 6 7 8
Goal:
0 1 2
3 4 5
6 7 8
Initial:
1 2 5
6 3 4
7 8 0
Initial Heuristic Value 8
Step: 1
Right   Left    Up      Down
8       7       8       8
7
left
1 2 5
6 3 4
7 0 8

Step: 2
Right   Left    Up      Down
8       6       7       7
6
left
1 2 5
6 3 4
0 7 8

Step: 3
Right   Left    Up      Down
7       6       5       6
5
up
1 2 5
0 3 4
6 7 8

Step: 4
Right   Left    Up      Down
4       5       5       6
4
right
1 2 5
3 0 4
6 7 8
```



```
Step: 5
Right   Left    Up      Down
3       5       4       5
3
right
1 2 5
3 4 0
6 7 8

Step: 6
Right   Left    Up      Down
3       4       2       4
2
up
1 2 0
3 4 5
6 7 8

Step: 7
Right   Left    Up      Down
2       1       2       3
1
left
1 0 2
3 4 5
6 7 8

Step: 8
Right   Left    Up      Down
2       0       1       2
0
left
0 1 2
3 4 5
6 7 8
Goal state is reached in 8 steps.
```

Observations:

- We observed that the Greedy Best First Search does not “Always” give an optimal solution.
- When the heuristic function does not decrease, it gets stuck in a loop.
- When the value of heuristic function is same for two or more states, any of those state is selected at random.
- For some specific inputs, we receive an optimal solution.