ITCS-6150 – Intelligent Systems

Project 1

Solving the 8-puzzle using A* search algorithm

Project Report

submitted by

Alaa Alharthi

Anisha Kakwani

Deep Prajapati

Heena Jain

At

University Of North Carolina at Charlotte

Problem Formulation:

8 puzzle problem: Here, 3 X 3 grid with 8 tiles is given where each tile has one number from 1 to 8 and one empty space with number 0. Our objective is to rearrange the tiles so that they are in the same order as given goal state. We can move the tiles horizontally and vertically (Left, Right, Above, Below).

A* Search: It is used for path traversal and path finding in problems by searching. It is one of the important algorithms in the domain of Informed Search Strategies. The main objective of this algorithm is to avoid expanding paths that are expensive. Therefore, this search algorithm is optimal. Also, limitation of greedy search is solved by this algorithm as it is using heuristic function as well as uniform cost value. The evaluation function used in A* search is f(n) = g(n) + h(n), where

g(n) = Cost to reach the node n from the source node

h(n) = Estimated cost from node n to goal

f(n)= Total estimated cost of the cheapest solution through node n.

Problem Elements:

Initial State: It is the state where the agent is in and from where it starts solving the puzzle. In the designed program, the user is supposed to provide the input initial state.

Goal State: It the final state in which the agent wants to be. In the designed program, the user enters the goal state.

State: A state is any arrangement of the cells of 8-puzzle that defines a particular physical configuration. In this case, the blank tile is swapped with other elements based on the rules to move from one state to another.

State Space: It is a set of all the possible states reachable from the initial state. In this problem structure, we get different possible arrangements of the digits and the blank tile in a 3x3 cell.

Possible Actions: There are 4 possible actions in the context of this problem i.e. up, down, left, and right for the blank (0) tile.

Formulation: A successor formulation of explored states is used which tracks the reachable state from any given state. Successors are decided based on two functions, heuristic function h(n) and cost function g(n). We have used two heuristic functions in this project: Manhattan distance heuristic and misplaced tiles heuristic.

Path: It is a sequence of actions.

Path Cost: It is a cost function that shows the measure of cost to arrive at a specific state. For this problem, the path cost to solution would become equal to g(n) eventually since h(n) will be 0 at goal state.

Step Cost: It is the cost of a single action step. For this problem structure, step cost = 1 for each move. It can also be interpreted as the depth level of a particular node.

Solution: It is a sequence of actions that we make from our initial state to the goal state.

Strategy: It is the method for node expansion. In A* search, the next node for expansion is selected based on the estimated cost of nodes that are obtained from the evaluation function. The node with the least estimated cost is expanded next. Strategies are assessed in terms of the following properties which are presented below with the true properties of A* search:

- Completeness - It will always find the solution unless there are infinitely many nodes with $f \leq f(G)$.
- Time complexity - Exponential in (relative error in h × length of solution)
- Space complexity - Proportional to the number of nodes generated as it keeps all nodes in memory.
- Optimality - A* will always find a least-cost solution.

Program Structure:

The overall flow of the program is to take as input an initial state and a goal state, the heuristic function user wishes to use by selecting 1 for Manhattan distance heuristic and 2 for Misplaced Tiles heuristic from the user. Based on the input provided, the A* search algorithm is performed to either obtain an optimal solution or return failure because output could not be found. The terminating condition that has been used in the program to handle invalid cases is an upper limit on the number of nodes generated. This limit is set at 3000. If a solution cannot be found before this limit, then the program states that output cannot be found. To generate the final solution, we have used backtracking from goal to source.

In our program, we have two classes. The first one is the Node class which is responsible for the configuration of the state structure. The second class is the Grid class which is responsible for the search process and its working depending on which heuristic function is selected by the user. We have used Python language for implementation of A* search using two different heuristics.

Node Class:

- **Variables:**
  - mat: matrix is used for reprinting the grid from 1 to 8.
  - gn: integer value that specify the depth of the node.
  - fn: integer value that specify the evaluation function of the current node.
  - parent_x: x location for the blank tile. Used to backtrack the path.

- parent_y: y location for the parent blank tile. Used to backtrack the path.
- matParent: parent matrix of current node. Used to remove the repeated nodes.

- **Functions:**
  - `def successor(self):`
    This function will return array of children of current node. It will find the location (x,y) of blank tile in current node and then generate 4 new locations for the blank tiles. After that it will use those locations to generate children nodes. This function will prevent the invalid tiles positions.

  - `def swapping (self, current_mat,x1,y1,x,y):`
    This function is responsible of getting a copy of current node to swapp ing it with new blank tile location. The attributes are the current matrix, x1 is new x blank location and y1 is new y blank location.

  - `def blank_tile(self,m,a):`
    This function is responsible of finding the blank tile location , where m is node matrix and a is the 0 , blank tile value.

Grid Class:

- **Variables:**
  - explored_list: list to keep all the explored nodes
  - frontier: list to keep the successors
  - no_of_nodes_generated: integer variable to keep track of the number of generated nodes at each iteration in search method.
- **Functions:**
  - `def read_input(self):`
    This function is responsible for taking the input from the user

  - `def matchWithParent(self,current_mat,parent_mat):`
    This function is responsible of checking whether the parent node is ge nerated again and prevent that if it happens. It is working to compare the child matrix with the current node parent matrix.

  - `def evaluation_fn(self,current,goal,heuristic):`
    This function is responsible for computing the f(n) value by adding the h(n) of the current node to g(n) of the current node. It calls the heuristic function first and do the addition later.

  - `def misplaced_tiles_heuristic(self,current,goal):`
    This function is responsible for computing h(n) based on misplaced til es heuristic. It uses two nested loops to compute the misplace tiles.

- `def manhattan_distance_heuristic(self,current,goal):`
  This function is responsible for computing h(n) based on manhattan di stance heuristic. It finds the distance by subtracting tile location in goal state from the tile location in the current state, then addition the two result to compute the final distance.

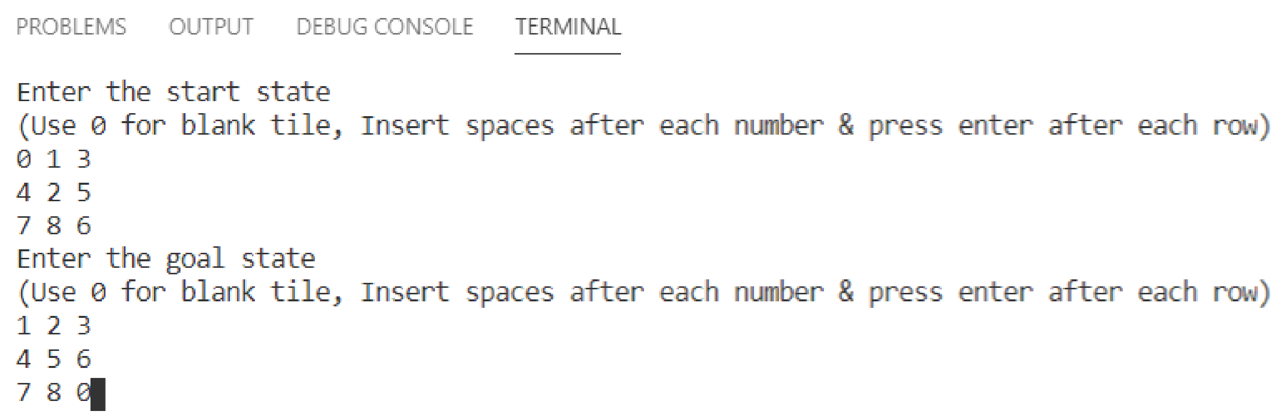
- `def find_tile_positions(self,state):`
  This is a secondary function used in the manhattan_distance_heuristic() to find the tiles locations.

- `def search(self):`
  This is a primary function which conduct the A* algorithm procedure. First it reads the start state and the goal sate from the user. Second, it computes the evaluation function of the start node. Third, it goes in a loop and it removes the node at the front and check if that node is the goal state. If not, it will generate the successor by using the successor() methods. Forth, it will check if node is not repeated, so it prevents adding it of the frontier list. The last thing to do is to store the explored node in explored list, sort the frontier based on the evaluation function, and continues in the loop until the solution is found or the frontier equal to zero.

- `def printSolutionPath(self,node,explored_list):`
  This function is responsible for printing the solution path, and it uses the parent x,y and g(n) to specify the path correctly

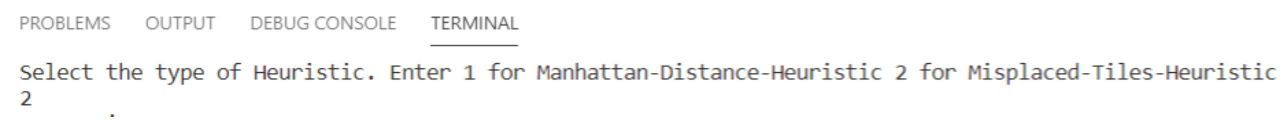
Procedure:

The program asks the user for the start node and the goal node:



```
Enter the start state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
0 1 3
4 2 5
7 8 6
Enter the goal state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
1 2 3
4 5 6
7 8 0
```

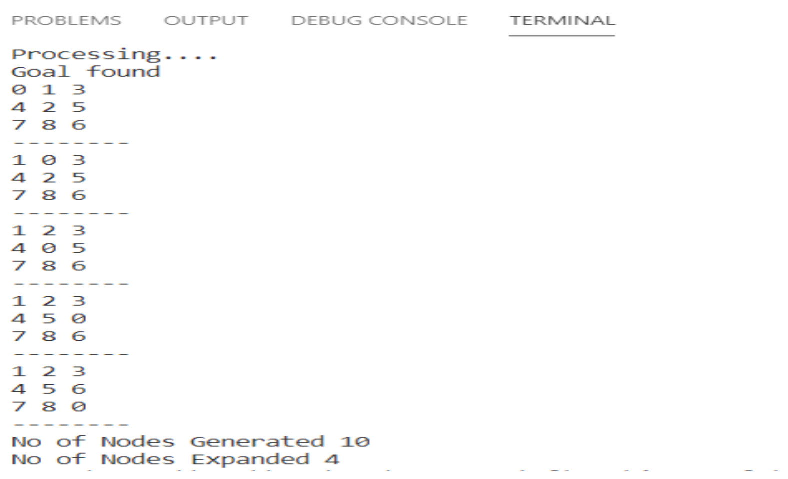
Then, ask the user which heuristic function he/she wants to use:



```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Select the type of Heuristic. Enter 1 for Manhattan-Distance-Heuristic 2 for Misplaced-Tiles-Heuristic
2
```

After that it performs A* search and then print the output path from the start to the goal state:



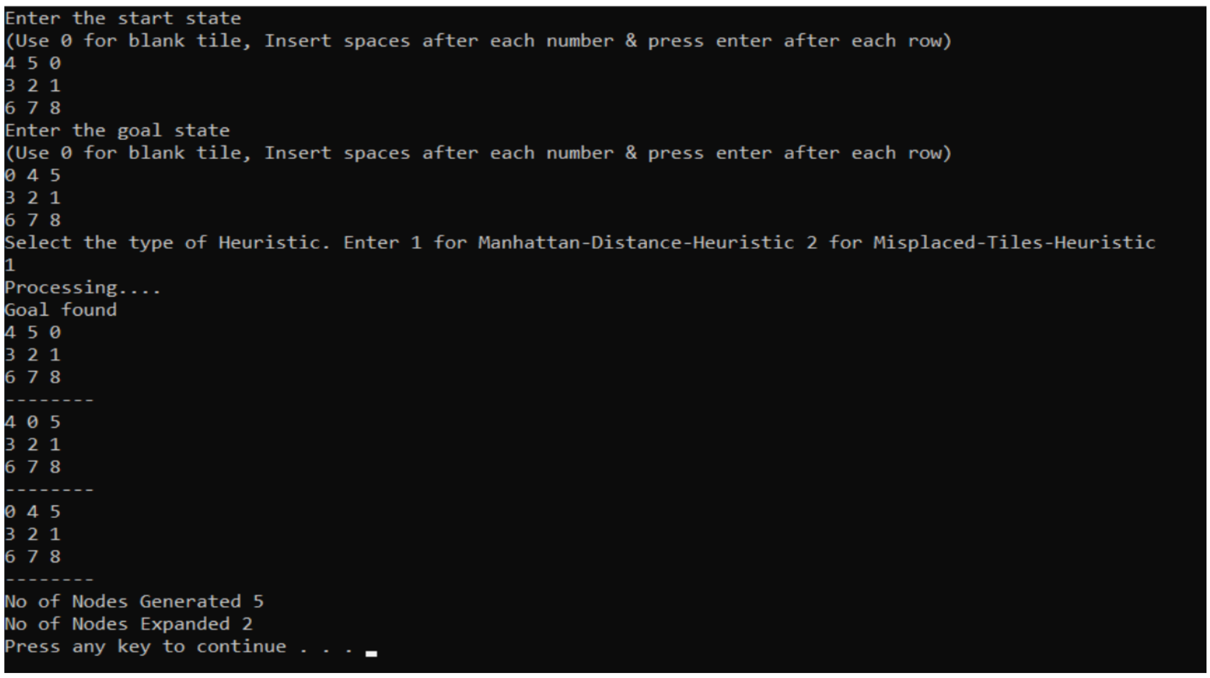
```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Processing....
Goal found
0 1 3
4 2 5
7 8 6
--------
1 0 3
4 2 5
7 8 6
--------
1 2 3
4 0 5
7 8 6
--------
1 2 3
4 5 0
7 8 6
--------
1 2 3
4 5 6
7 8 0
--------
No of Nodes Generated 10
No of Nodes Expanded 4
```

Analysis of 6 Input and output :

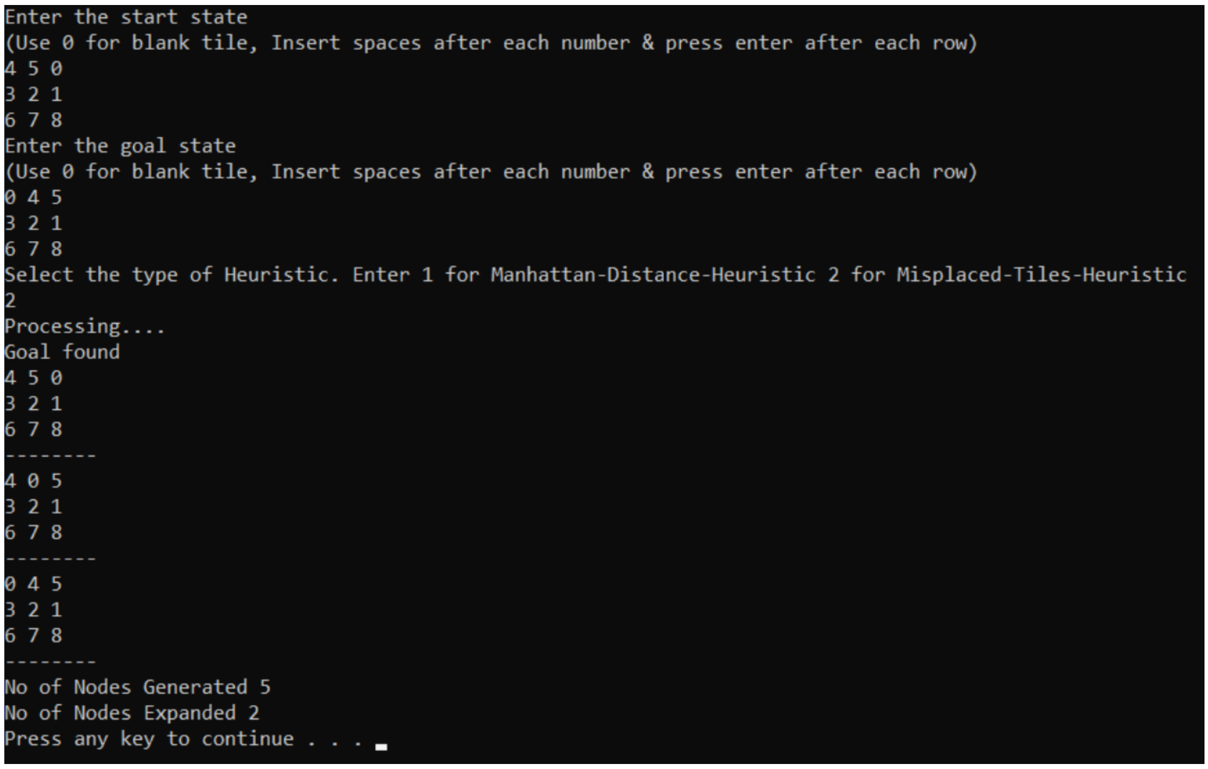
- Input 1: Below are the screenshots of the output for input instance 1. The input and goal states can be seen at the top. The number of nodes generated and expanded are shown along with the solution path.

Manhattan Distance Heuristic:



```
Enter the start state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
4 5 0
3 2 1
6 7 8
Enter the goal state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
0 4 5
3 2 1
6 7 8
Select the type of Heuristic. Enter 1 for Manhattan-Distance-Heuristic 2 for Misplaced-Tiles-Heuristic
1
Processing....
Goal found
4 5 0
3 2 1
6 7 8
--------
4 0 5
3 2 1
6 7 8
--------
0 4 5
3 2 1
6 7 8
--------
No of Nodes Generated 5
No of Nodes Expanded 2
Press any key to continue . . .
```

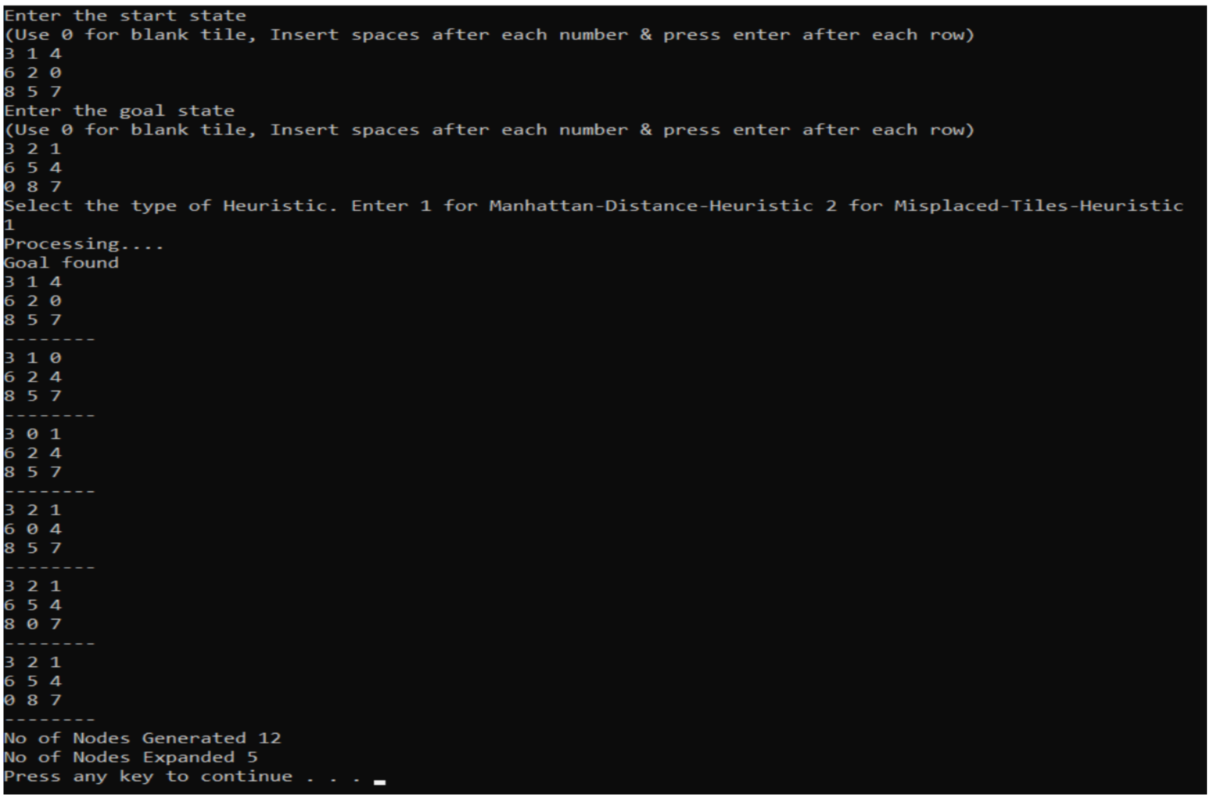
Misplaced Tiles Heuristic:



```
Enter the start state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
4 5 0
3 2 1
6 7 8
Enter the goal state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
0 4 5
3 2 1
6 7 8
Select the type of Heuristic. Enter 1 for Manhattan-Distance-Heuristic 2 for Misplaced-Tiles-Heuristic
2
Processing....
Goal found
4 5 0
3 2 1
6 7 8
--------
4 0 5
3 2 1
6 7 8
--------
0 4 5
3 2 1
6 7 8
--------
No of Nodes Generated 5
No of Nodes Expanded 2
Press any key to continue . . .
```

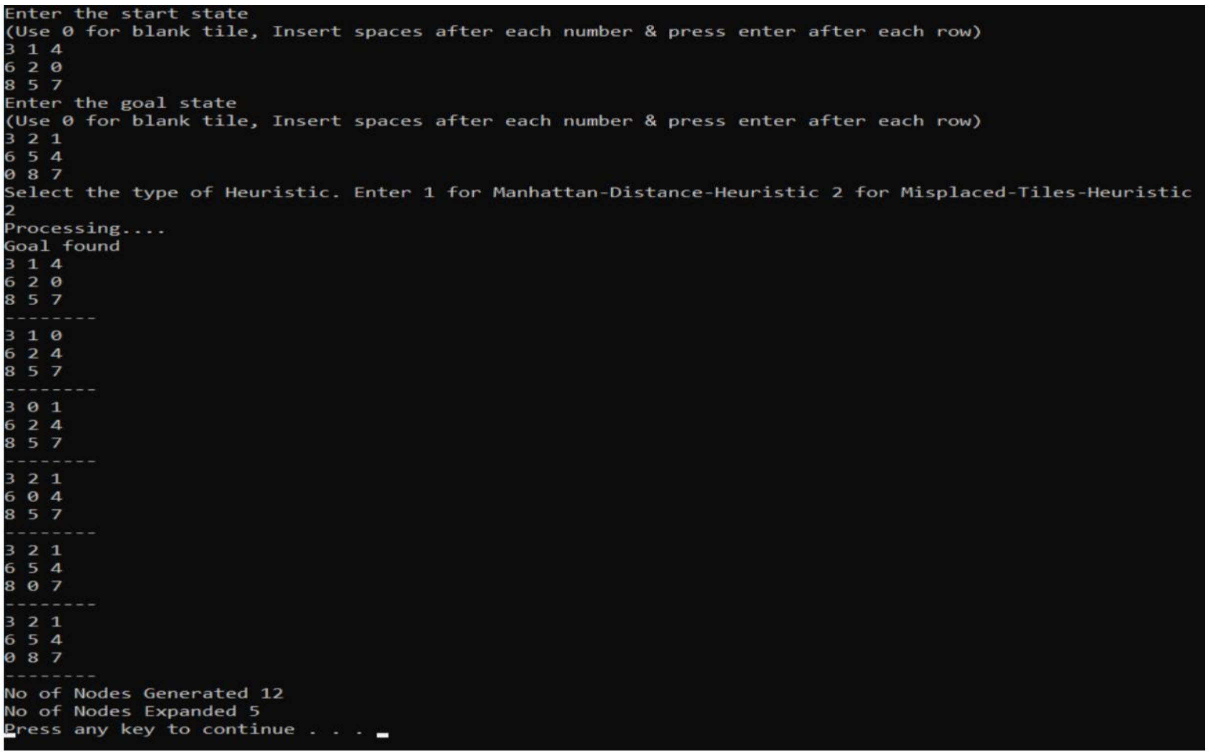
- Input 2: Below are the screenshots of the output for input instance 2. The input and goal states can be seen at the top. The number of nodes generated and expanded are shown along with the solution path.

Manhattan Distance Heuristic:



```
Enter the start state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
3 1 4
6 2 0
8 5 7
Enter the goal state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
3 2 1
6 5 4
0 8 7
Select the type of Heuristic. Enter 1 for Manhattan-Distance-Heuristic 2 for Misplaced-Tiles-Heuristic
1
Processing....
Goal found
3 1 4
6 2 0
8 5 7
--------
3 1 0
6 2 4
8 5 7
--------
3 0 1
6 2 4
8 5 7
--------
3 2 1
6 0 4
8 5 7
--------
3 2 1
6 5 4
8 0 7
--------
3 2 1
6 5 4
0 8 7
--------
No of Nodes Generated 12
No of Nodes Expanded 5
Press any key to continue . . .
```

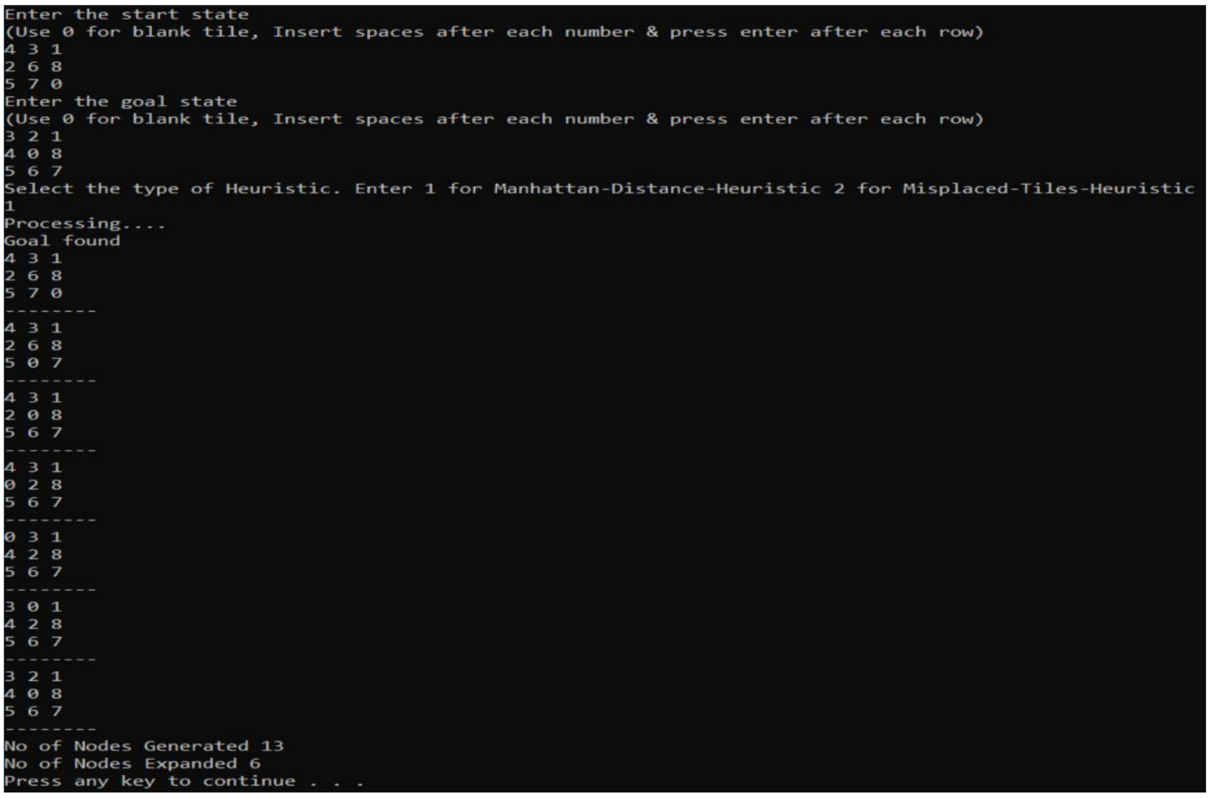
Misplaced Tiles Heuristic:



```
Enter the start state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
3 1 4
6 2 0
8 5 7
Enter the goal state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
3 2 1
6 5 4
0 8 7
Select the type of Heuristic. Enter 1 for Manhattan-Distance-Heuristic 2 for Misplaced-Tiles-Heuristic
2
Processing....
Goal found
3 1 4
6 2 0
8 5 7
--------
3 1 0
6 2 4
8 5 7
--------
3 0 1
6 2 4
8 5 7
--------
3 2 1
6 0 4
8 5 7
--------
3 2 1
6 5 4
8 0 7
--------
3 2 1
6 5 4
0 8 7
--------
No of Nodes Generated 12
No of Nodes Expanded 5
Press any key to continue . . .
```

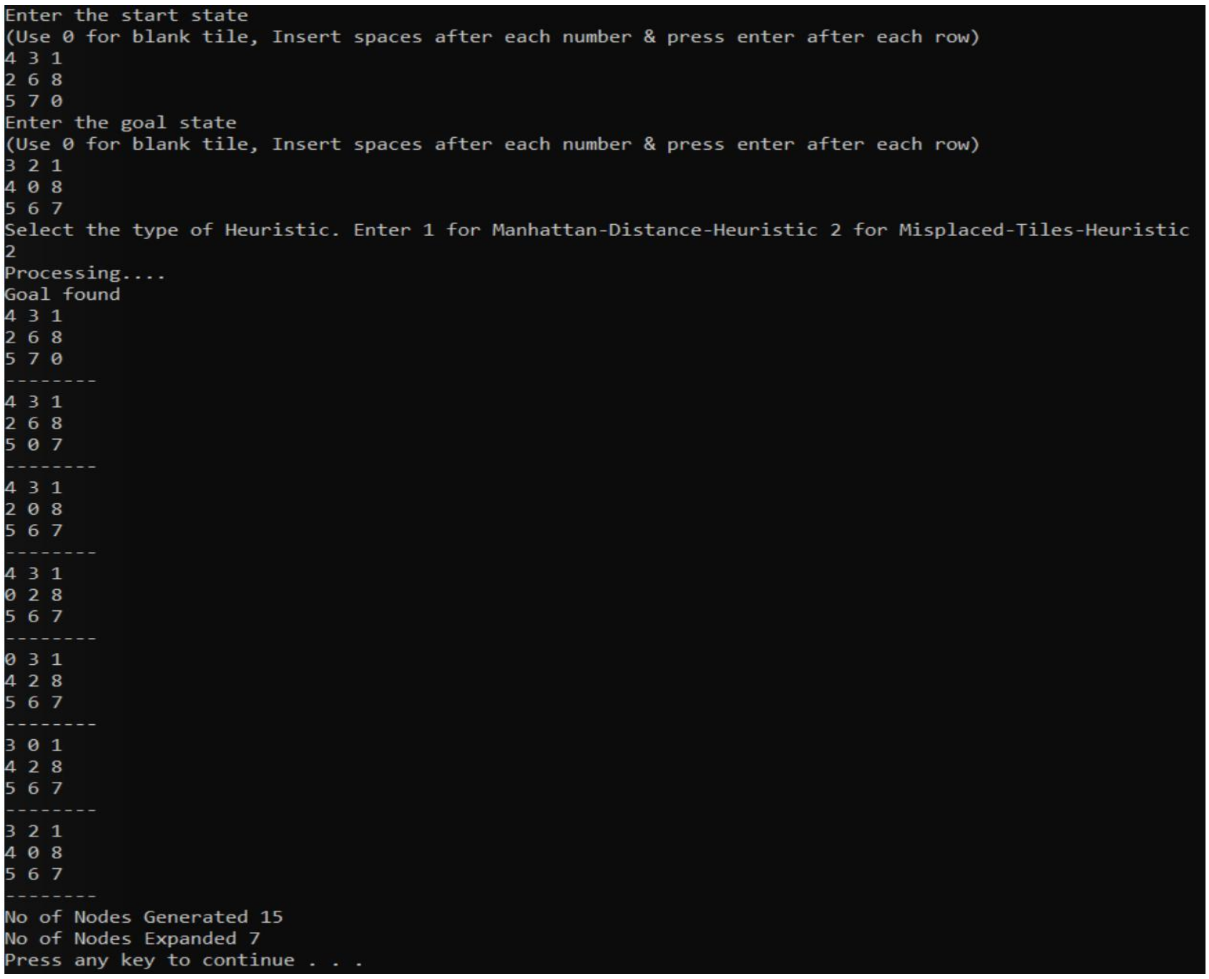
- Input 3: Below are the screenshots of the output for input instance 3. The input and goal states can be seen at the top. The number of nodes generated and expanded are shown along with the solution path.

Manhattan Distance Heuristic:



```
Enter the start state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
4 3 1
2 6 8
5 7 0
Enter the goal state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
3 2 1
4 0 8
5 6 7
Select the type of Heuristic. Enter 1 for Manhattan-Distance-Heuristic 2 for Misplaced-Tiles-Heuristic
1
Processing....
Goal found
4 3 1
2 6 8
5 7 0
--------
4 3 1
2 6 8
5 0 7
--------
4 3 1
2 0 8
5 6 7
--------
4 3 1
0 2 8
5 6 7
--------
0 3 1
4 2 8
5 6 7
--------
3 0 1
4 2 8
5 6 7
--------
3 2 1
4 0 8
5 6 7
--------
No of Nodes Generated 13
No of Nodes Expanded 6
Press any key to continue . . .
```

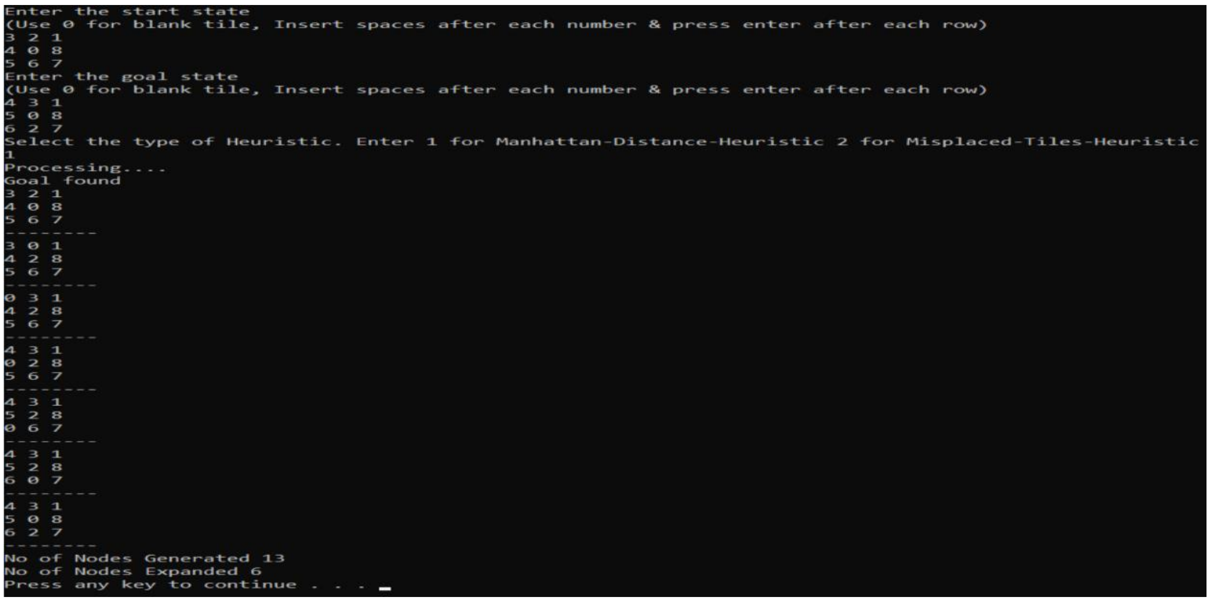
Misplaced Tiles Heuristic:



```
Enter the start state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
4 3 1
2 6 8
5 7 0
Enter the goal state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
3 2 1
4 0 8
5 6 7
Select the type of Heuristic. Enter 1 for Manhattan-Distance-Heuristic 2 for Misplaced-Tiles-Heuristic
2
Processing....
Goal found
4 3 1
2 6 8
5 7 0
--------
4 3 1
2 6 8
5 0 7
--------
4 3 1
2 0 8
5 6 7
--------
4 3 1
0 2 8
5 6 7
--------
0 3 1
4 2 8
5 6 7
--------
3 0 1
4 2 8
5 6 7
--------
3 2 1
4 0 8
5 6 7
--------
No of Nodes Generated 15
No of Nodes Expanded 7
Press any key to continue . . .
```

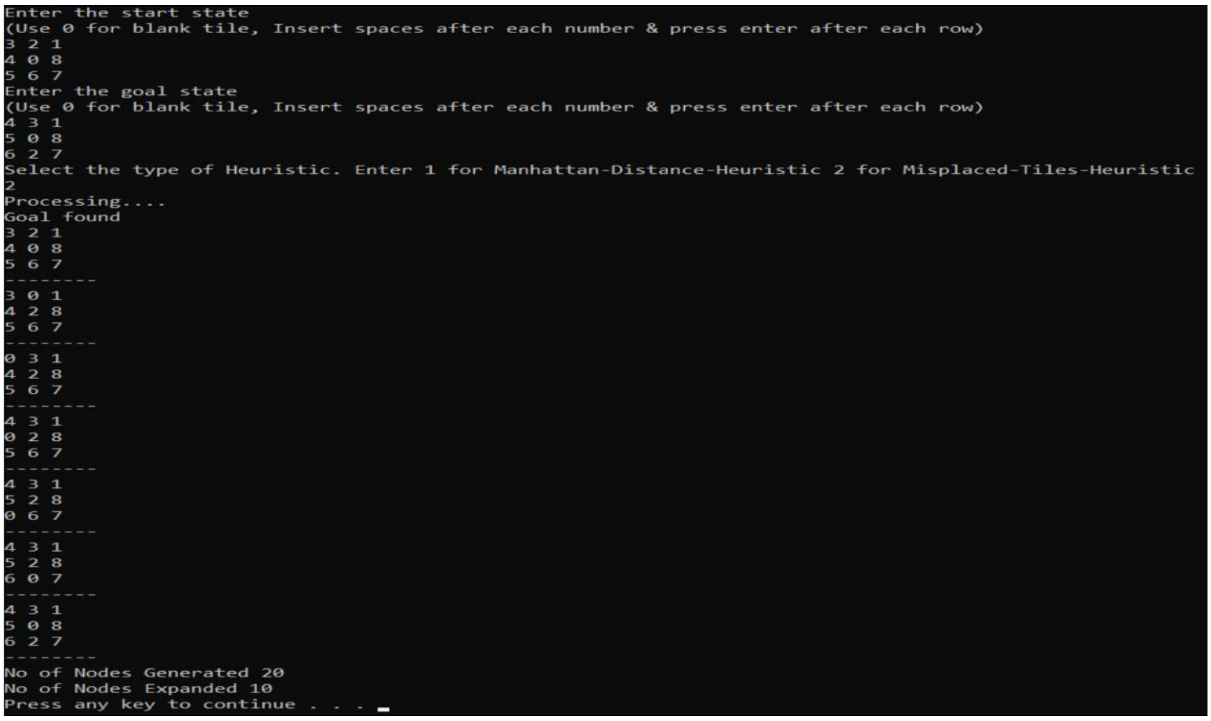
- Input 4: Below are the screenshots of the output for input instance 4. The input and goal states can be seen at the top. The number of nodes generated and expanded are shown along with the solution path.

Manhattan Distance Heuristic:



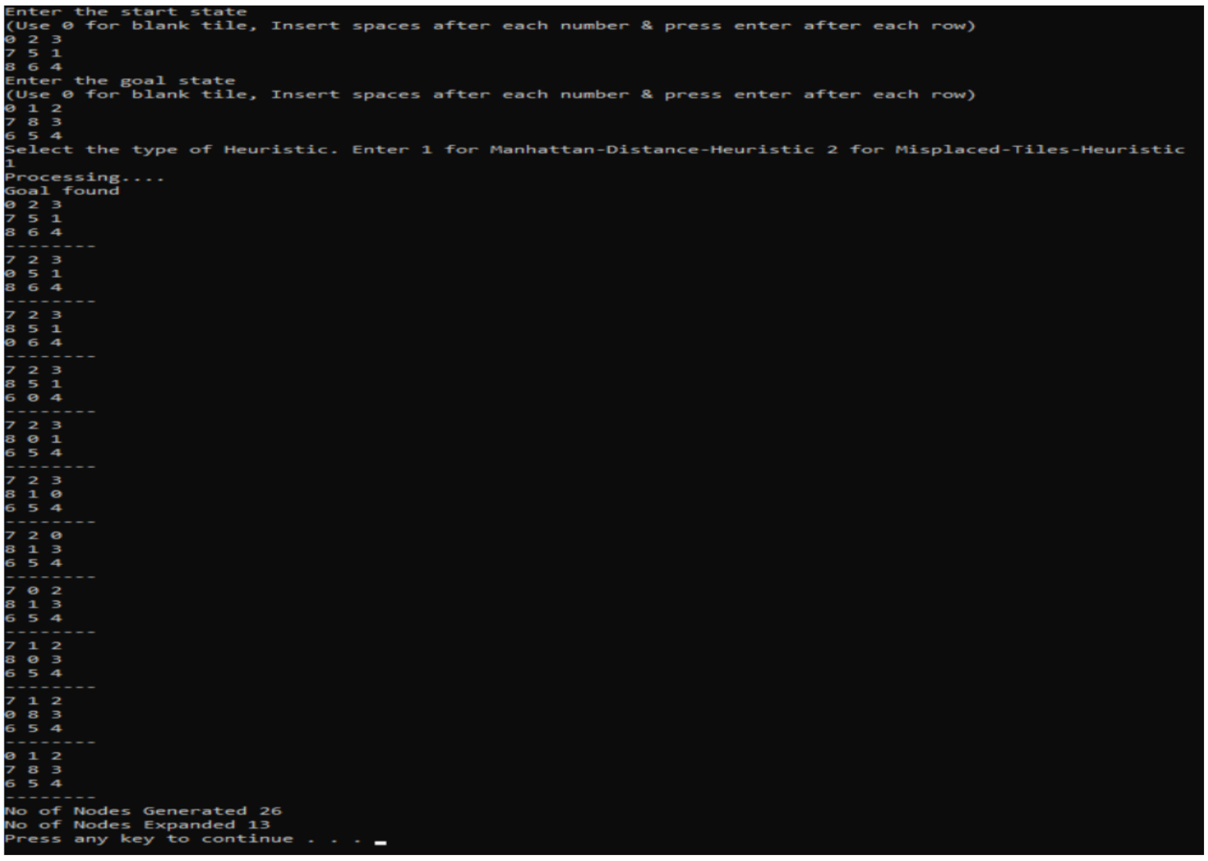
```
Enter the start state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
3 2 1
4 0 8
5 6 7
Enter the goal state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
4 3 1
5 0 8
6 2 7
Select the type of Heuristic. Enter 1 for Manhattan-Distance-Heuristic 2 for Misplaced-Tiles-Heuristic
1
Processing....
Goal found
3 2 1
4 0 8
5 6 7
--------
3 0 1
4 2 8
5 6 7
--------
0 3 1
4 2 8
5 6 7
--------
4 3 1
0 2 8
5 6 7
--------
4 3 1
5 2 8
0 6 7
--------
4 3 1
5 2 8
6 0 7
--------
4 3 1
5 0 8
6 2 7
--------
No of Nodes Generated 13
No of Nodes Expanded 6
Press any key to continue . . .
```

Misplaced Tiles Heuristic:

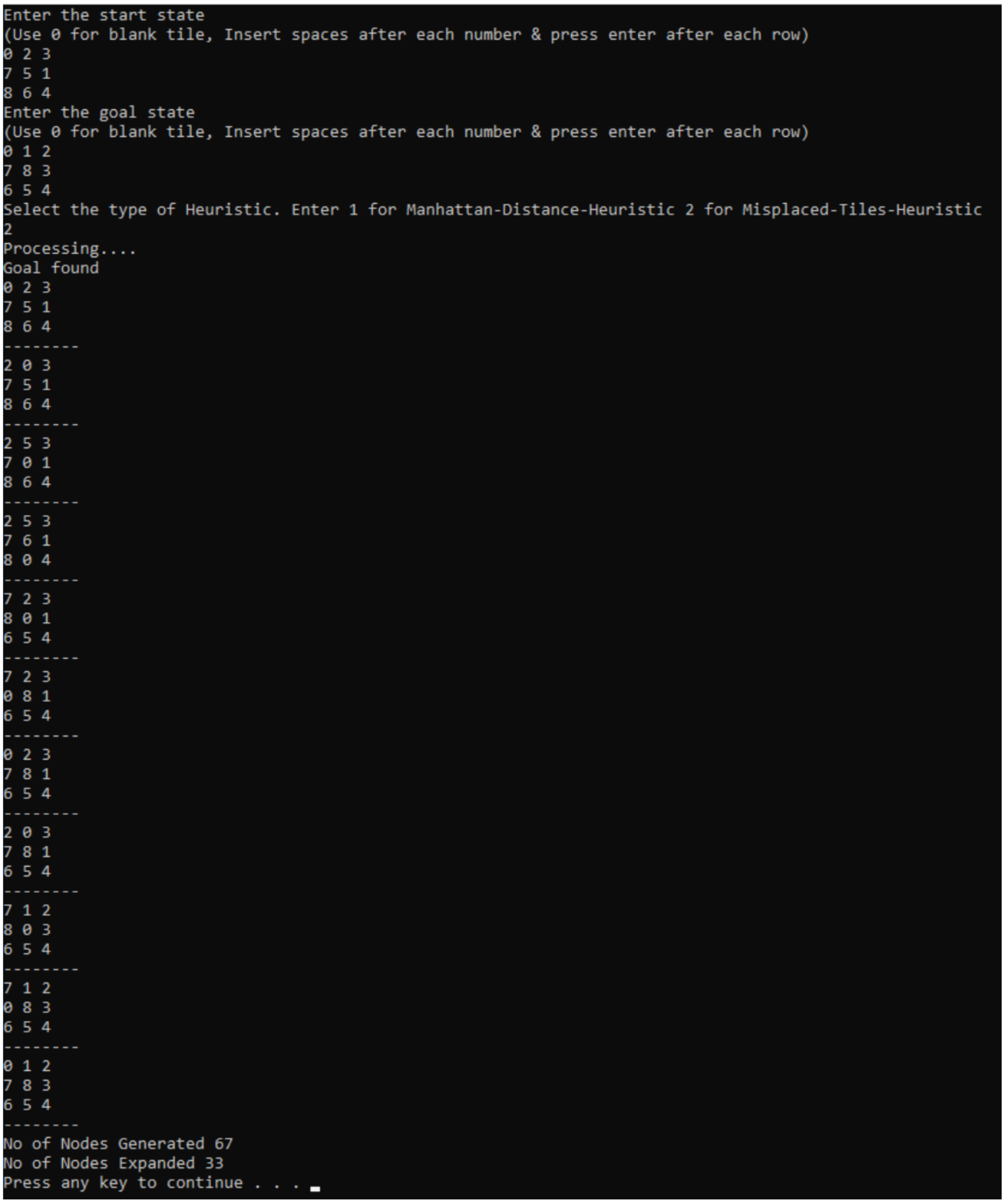

```
Enter the start state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
3 2 1
4 0 8
5 6 7
Enter the goal state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
4 3 1
5 0 8
6 2 7
Select the type of Heuristic. Enter 1 for Manhattan-Distance-Heuristic 2 for Misplaced-Tiles-Heuristic
2
Processing....
Goal found
3 2 1
4 0 8
5 6 7
--------
3 0 1
4 2 8
5 6 7
--------
0 3 1
4 2 8
5 6 7
--------
4 3 1
0 2 8
5 6 7
--------
4 3 1
5 2 8
0 6 7
--------
4 3 1
5 2 8
6 0 7
--------
4 3 1
5 0 8
6 2 7
--------
No of Nodes Generated 20
No of Nodes Expanded 10
Press any key to continue . . .
```

- Input 5: Below are the screenshots of the output for input instance 5. The input and goal states can be seen at the top. The number of nodes generated and expanded are shown along with the solution path.

Manhattan Distance Heuristic:


```
Enter the start state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
0 2 3
7 5 1
8 6 4
Enter the goal state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
0 1 2
7 8 3
6 5 4
Select the type of Heuristic. Enter 1 for Manhattan-Distance-Heuristic 2 for Misplaced-Tiles-Heuristic
1
Processing....
Goal found
0 2 3
7 5 1
8 6 4
--------
7 2 3
0 5 1
8 6 4
--------
7 2 3
8 5 1
0 6 4
--------
7 2 3
8 5 1
6 0 4
--------
7 2 3
8 0 1
6 5 4
--------
7 2 3
8 1 0
6 5 4
--------
7 2 0
8 1 3
6 5 4
--------
7 0 2
8 1 3
6 5 4
--------
7 1 2
8 0 3
6 5 4
--------
7 1 2
0 8 3
6 5 4
--------
0 1 2
7 8 3
6 5 4
--------
No of Nodes Generated 26
No of Nodes Expanded 13
Press any key to continue . . .
```

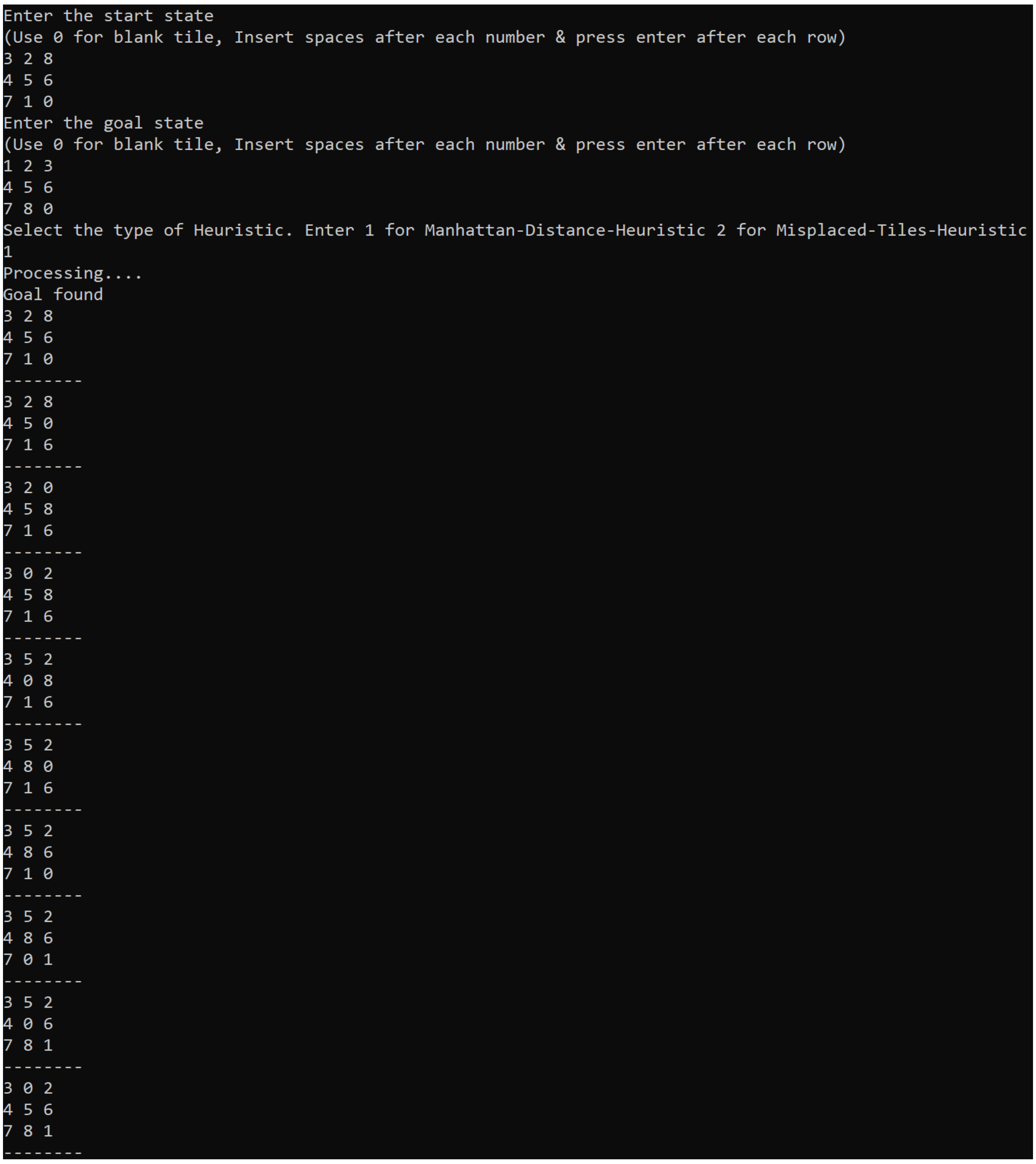
Misplaced Tiles Heuristic:



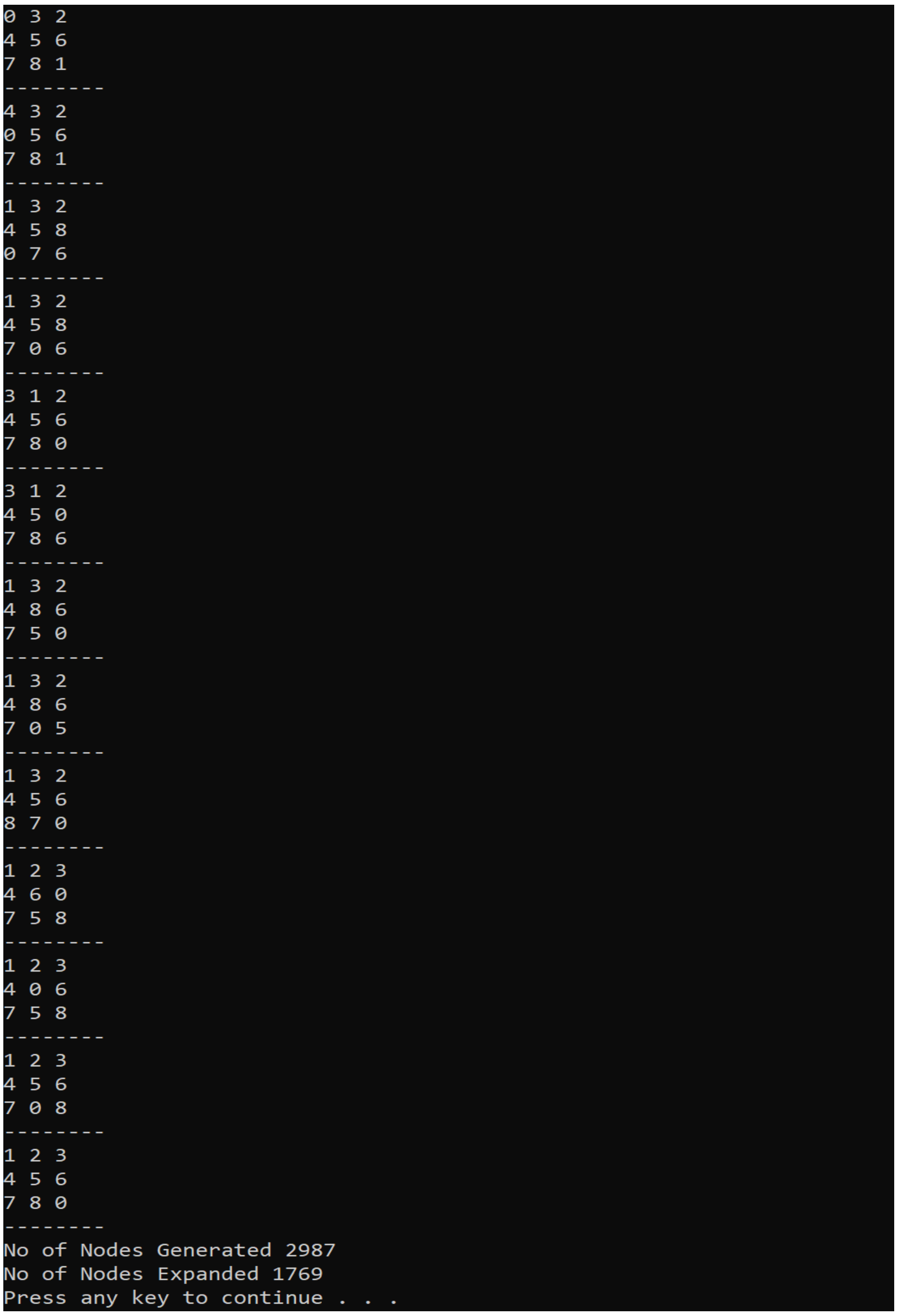
```
Enter the start state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
0 2 3
7 5 1
8 6 4
Enter the goal state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
0 1 2
7 8 3
6 5 4
Select the type of Heuristic. Enter 1 for Manhattan-Distance-Heuristic 2 for Misplaced-Tiles-Heuristic
2
Processing....
Goal found
0 2 3
7 5 1
8 6 4
--------
2 0 3
7 5 1
8 6 4
--------
2 5 3
7 0 1
8 6 4
--------
2 5 3
7 6 1
8 0 4
--------
7 2 3
8 0 1
6 5 4
--------
7 2 3
0 8 1
6 5 4
--------
0 2 3
7 8 1
6 5 4
--------
2 0 3
7 8 1
6 5 4
--------
7 1 2
8 0 3
6 5 4
--------
7 1 2
0 8 3
6 5 4
--------
0 1 2
7 8 3
6 5 4
--------
No of Nodes Generated 67
No of Nodes Expanded 33
Press any key to continue . . .
```

- Input 6: Below are the screenshots of the output for input instance 6. The input and goal states can be seen at the top. The number of nodes generated and expanded are shown along with the solution path. The output screen is split into two screenshots because of the large number of nodes in the path.

Manhattan Distance Heuristic:

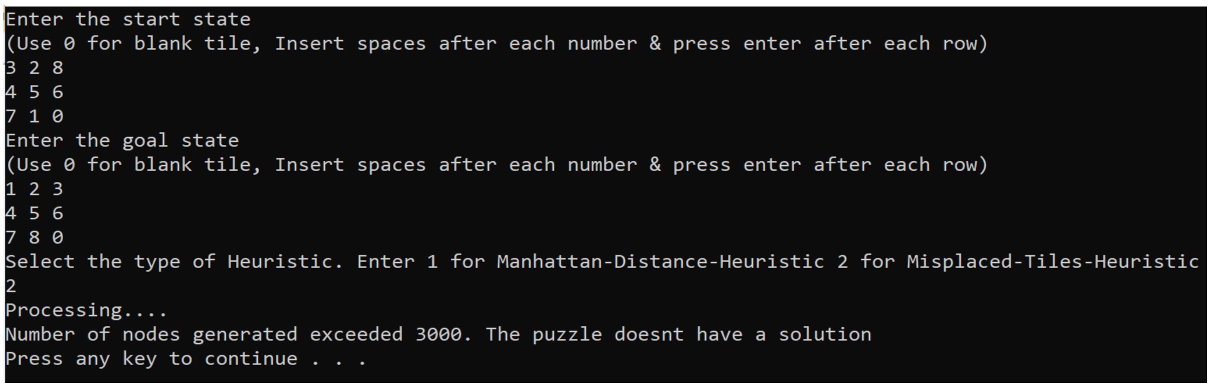


```
Enter the start state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
3 2 8
4 5 6
7 1 0
Enter the goal state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
1 2 3
4 5 6
7 8 0
Select the type of Heuristic. Enter 1 for Manhattan-Distance-Heuristic 2 for Misplaced-Tiles-Heuristic
1
Processing....
Goal found
3 2 8
4 5 6
7 1 0
--------
3 2 8
4 5 0
7 1 6
--------
3 2 0
4 5 8
7 1 6
--------
3 0 2
4 5 8
7 1 6
--------
3 5 2
4 0 8
7 1 6
--------
3 5 2
4 8 0
7 1 6
--------
3 5 2
4 8 6
7 1 0
--------
3 5 2
4 8 6
7 0 1
--------
3 5 2
4 0 6
7 8 1
--------
3 0 2
4 5 6
7 8 1
--------
```



```
0 3 2
4 5 6
7 8 1
--------
4 3 2
0 5 6
7 8 1
--------
1 3 2
4 5 8
0 7 6
--------
1 3 2
4 5 8
7 0 6
--------
3 1 2
4 5 6
7 8 0
--------
3 1 2
4 5 0
7 8 6
--------
1 3 2
4 8 6
7 5 0
--------
1 3 2
4 8 6
7 0 5
--------
1 3 2
4 5 6
8 7 0
--------
1 2 3
4 6 0
7 5 8
--------
1 2 3
4 0 6
7 5 8
--------
1 2 3
4 5 6
7 0 8
--------
1 2 3
4 5 6
7 8 0
--------
No of Nodes Generated 2987
No of Nodes Expanded 1769
Press any key to continue . . .
```

Misplaced Tiles Heuristic:



```
Enter the start state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
3 2 8
4 5 6
7 1 0
Enter the goal state
(Use 0 for blank tile, Insert spaces after each number & press enter after each row)
1 2 3
4 5 6
7 8 0
Select the type of Heuristic. Enter 1 for Manhattan-Distance-Heuristic 2 for Misplaced-Tiles-Heuristic
2
Processing....
Number of nodes generated exceeded 3000. The puzzle doesnt have a solution
Press any key to continue . . .
```

Summary of Input and Output cases:

Input Instance #	Approach	Initial State	Goal State	No. of Generated Nodes	No. of Expanded Nodes
1	Manhattan	4 5 0 3 2 1 6 7 8	0 4 5 3 2 1 6 7 8	5	2
	Misplaced Tiles			5	2
2	Manhattan	3 1 4 6 2 0 8 5 7	3 2 1 6 5 4 0 8 7	12	5
	Misplaced Tiles			12	5
3	Manhattan	4 3 1 2 6 8 5 7 0	3 2 1 4 0 8 5 6 7	12	5
	Misplaced Tiles			15	7
4	Manhattan	3 2 1 4 0 8 5 6 7	4 3 1 5 0 8 6 2 7	13	6
	Misplaced			20	10

5	Manhattan	0 2 3 7 5 1 8 6 4	0 1 2 7 8 3 6 5 4	26	13
	Misplaced Tiles			67	33
6	Manhattan	3 2 8 4 5 6 7 1 0	1 2 3 4 5 6 7 8 0	2987	1769
	Misplaced Tiles			3000	No solution

Table 1

Summary Analysis:

We identified 6 input instances for each of the two heuristic functions, namely, Manhattan Distance and Misplaced Tiles heuristic functions. For this project, we define a problem as invalid if it requires more than 3000 nodes to be generated before providing a solution. We decided this limit based on our dataset of input instances and took the maximum number of nodes that we generated for any problem before we found the solution. The results can be seen in Table 1. The Manhattan Distance heuristic function was able to solve all six instances within the specified and hence never returned a failure. The maximum number of nodes generated by this heuristic were 2987 for instance 6. The Misplaced Tiles heuristic, on the other hand, returned a failure for input instance 6 and returned an optimal path for all other instances. Overall, we found that the Manhattan Distance heuristic performs better with A* search algorithm than Misplaced Tiles heuristic because it generates and expands lesser number of nodes as compared to the latter and is, therefore, faster.