Project 1

Aravind Kumar Reddy Yempada
ITCS 6150
800877797

Program Details

The program has been developed in C# using the Visual Studio IDE. It contains three .cs files

- Program.cs
- AStarProgram.cs
- PriorityQueue.cs

Program.cs contains the Main() method from which the execution starts. Main() accepts the input from user and stores them in startNode and goalNode. It then creates an object of AStarProgram class and calls the AStar method to find the solution. It then displays the solution, number of nodes in the solution path, number of nodes generated and number of nodes expanded.

The Node class represents a state of 8-puzzle problem, it is implemented in the Program.cs. Node class contains the following properties and methods in it:

- State– represents a state
- Parent – reference of the parent node
- Pathcost –the cost required to reach current node from start node
- Stepcost – cost required to reach current node from parent node
- Priority – stores the f-cost and used in building priority queue
- ChildrenNodes() – Method to generate the successor nodes for the current node
- MoveOperation() – Method to move the blank space in a state to a desired position
- HCost() – calculates the heuristic cost which is the Manhattan distance between the start node and goal node.
- GCost() – calculates the path cost
- PrintState() – prints state on a console
- Equals() – compares two nodes and returns true if states are equal

The PriorityQueue.cs contains the implementation of priority queue. A priority queue is used to store the nodes generated by the A* algorithm and to remove the node that has the least f(n).

Node.Priority property is used to maintain the priority queue. This class has three methods to perform operations on priority queue such as insertion, deletion of first element and deletion of element at a required position.

AStarProgram.cs has the implementation of A* star algorithm given in the class handout with a modification to remove all the repeated nodes. The pseudo code is below:

```
1. Make a node INIT for the initial state;
2. Set the queue NODES to contain INIT;
3. Loop:
     if NODES is empty then report failure and
exit loop
     else
       begin
      move BESTNODE (which has the lowest f')
from the           front of NODES;
         if BESTNODE is the goal node then
            return BESTNODE and exit loop /**a
solution**/
         else
           begin
             generate the successor nodes of
BESTNODE;

             for each SUCCESSOR do
             begin

             /**discard SUCCESSOR if the state is
             the same as one**/
             /**of its ancestors', assuming no
          negative path cost**/

               if SUCCESSOR's state appears in an
                    ancestor node then
                  discard SUCCESSOR
                else
                begin

                  /**compute the path cost**/
                  g(SUCCESSOR) = g(BESTNODE)
                   + the       step cost from
                BESTNODE;
```

```
                  /**avoid repeated states in
                 the queue NODES**/
                 if SUCCESSOR's state is in
                 node OLD in NODES then
                   if g(OLD) > g(SUCCESSOR)
                 then
                   begin
                  remove OLD from NODES and
                        discard it;
                     set the SUCCESSOR's parent
                 link to BESTNODE;

                  f'(SUCCESSOR) = g(SUCCESSOR)
                  + h'(SUCCESSOR);
                 insert SUCCESSOR to NODES
                 based on f'
                   end
                   else discard SUCCESSOR
              end /**if not in ancestor's**/
          end; /**for**/
      end; /**if not goal**/
end; /**if not failure**/
```

Source: http://coitweb.uncc.edu/~xiao/itcs6150-8150/astar_handout.txt

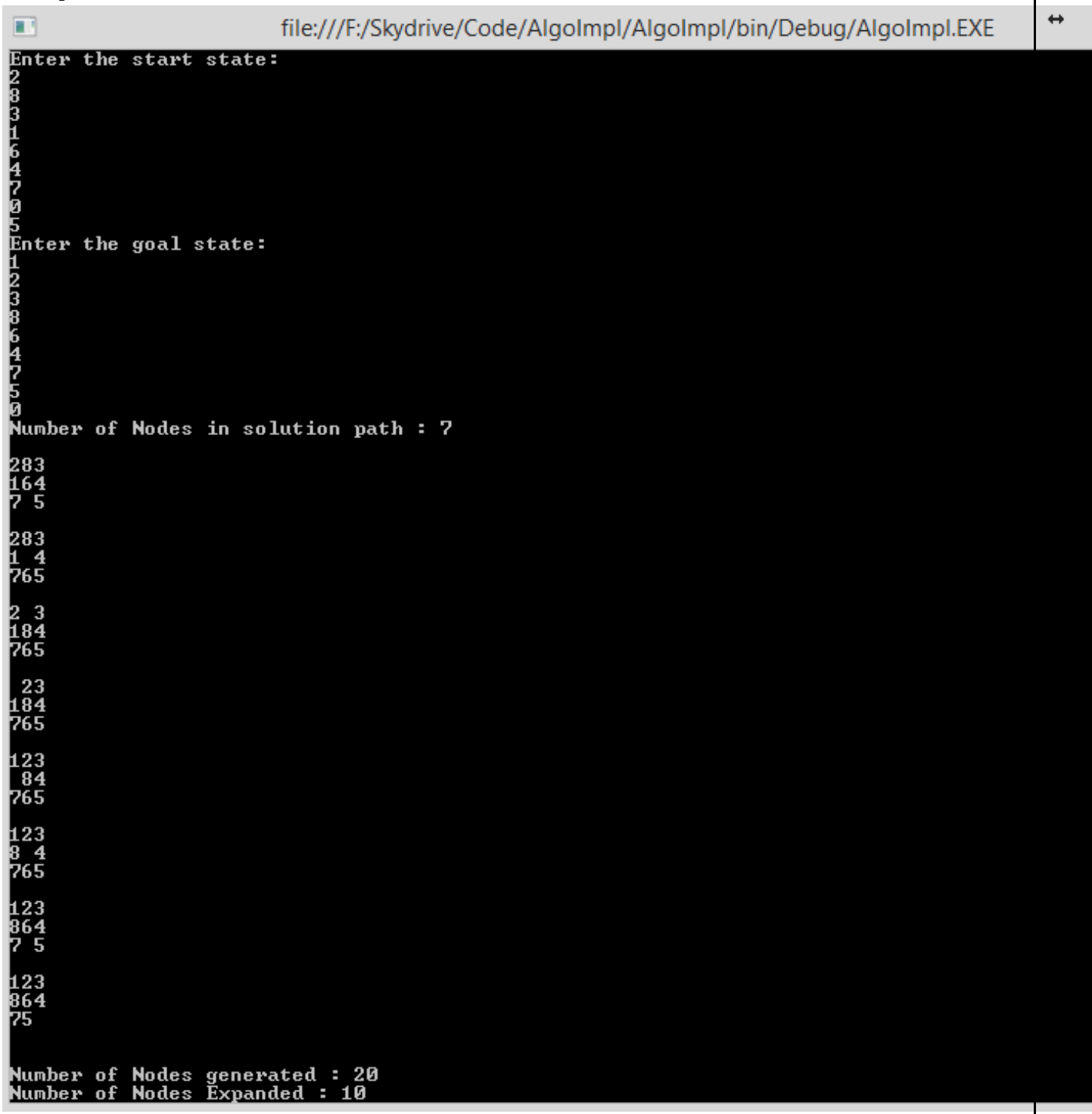
Following three global variables are used in the program:

- puzzleSize - To change the size of the puzzle as needed, it's value is 3 for this problem
- noOfGeneratedNodes – keeps track of number of nodes generated by the A* algorithm
- noOfExpandedNodes – keeps track of number of nodes expanded by the A* algorithm

On execution the program asks for user to enter the start node and goal node. The program accepts only digits 0-8 and repeated values are not accepted. On execution the program displays the number of nodes in solution path, each state in the path, number of nodes generated and number of nodes expanded.

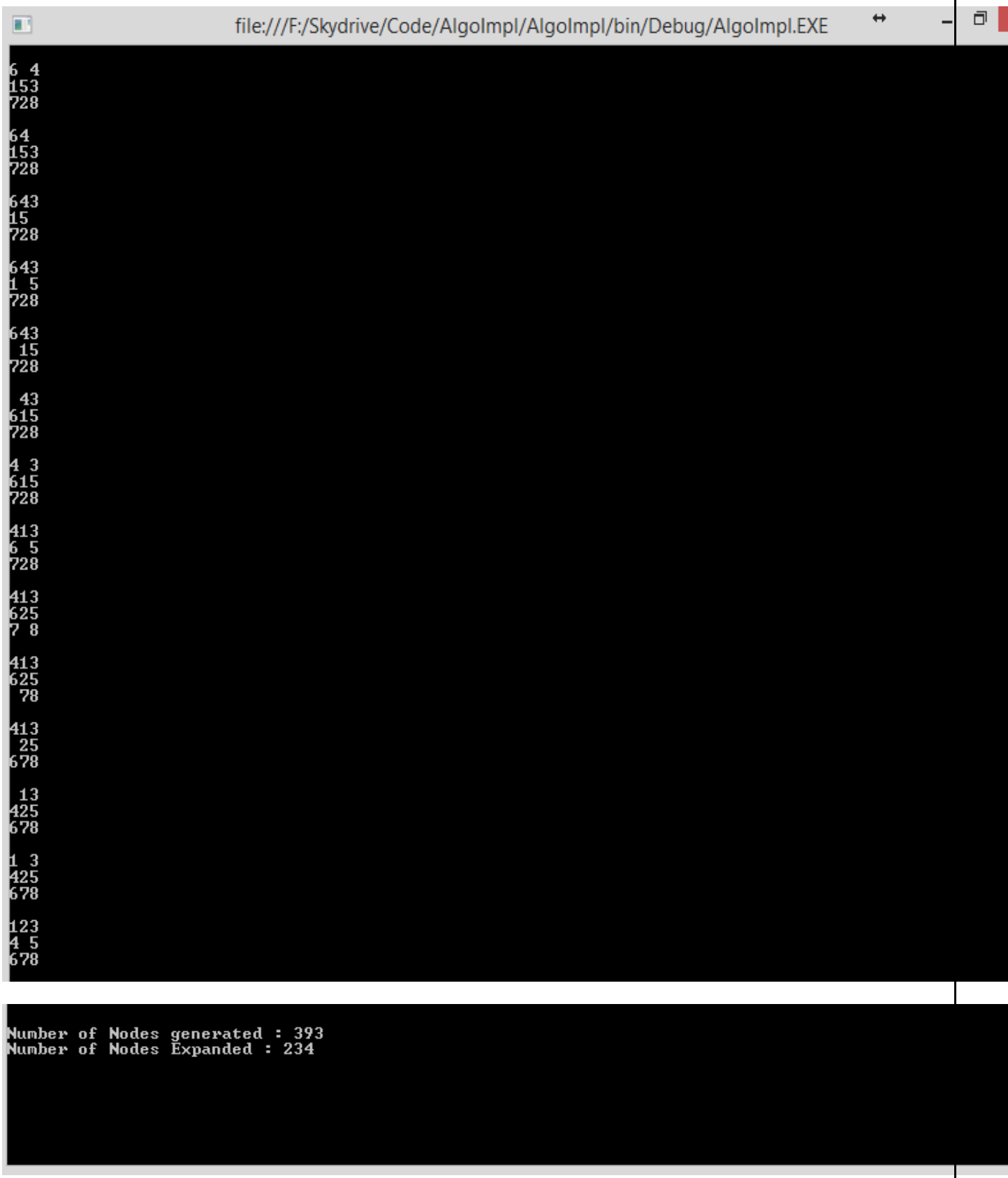
Execution Screenshots

Example I

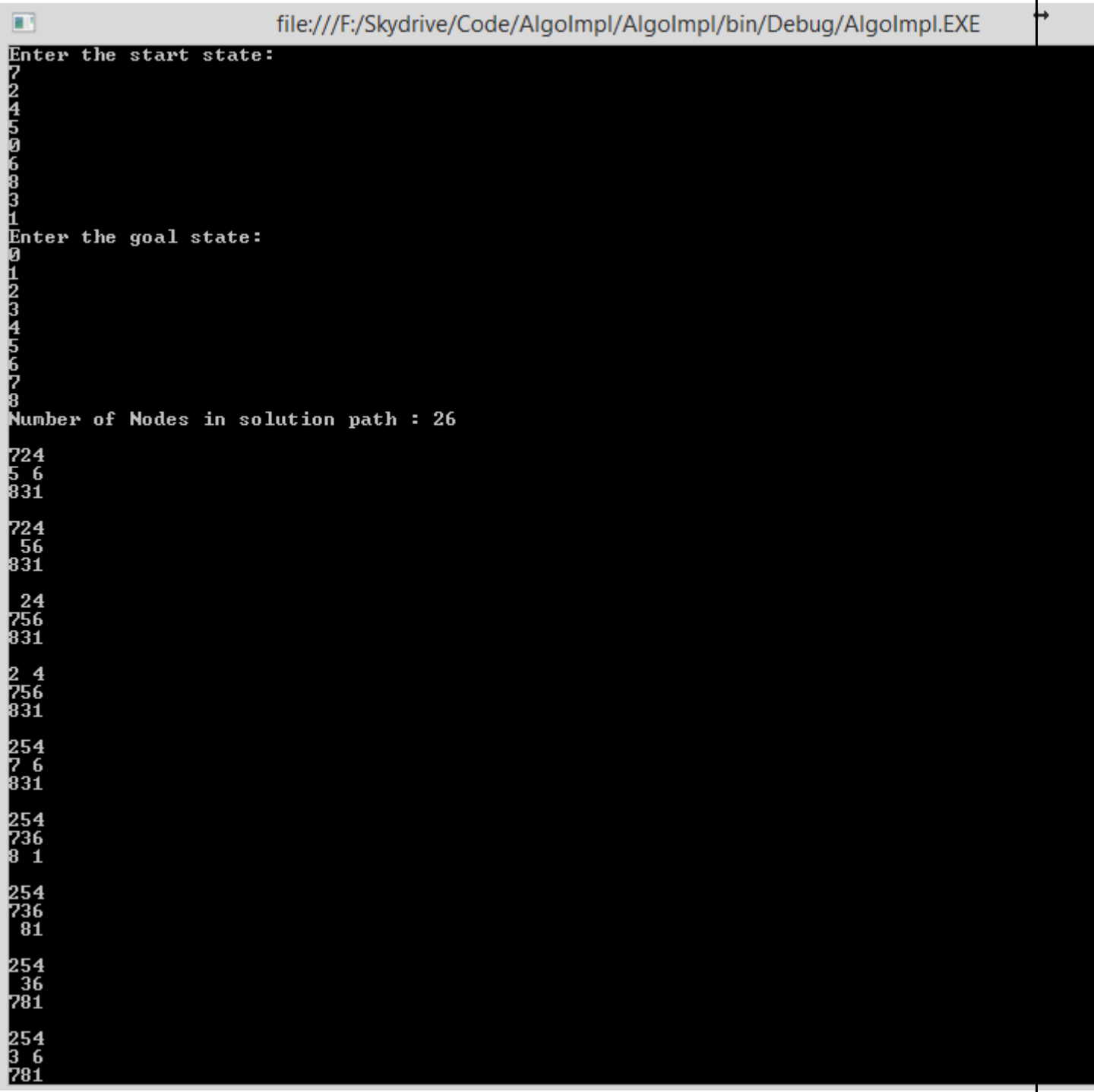


Example II

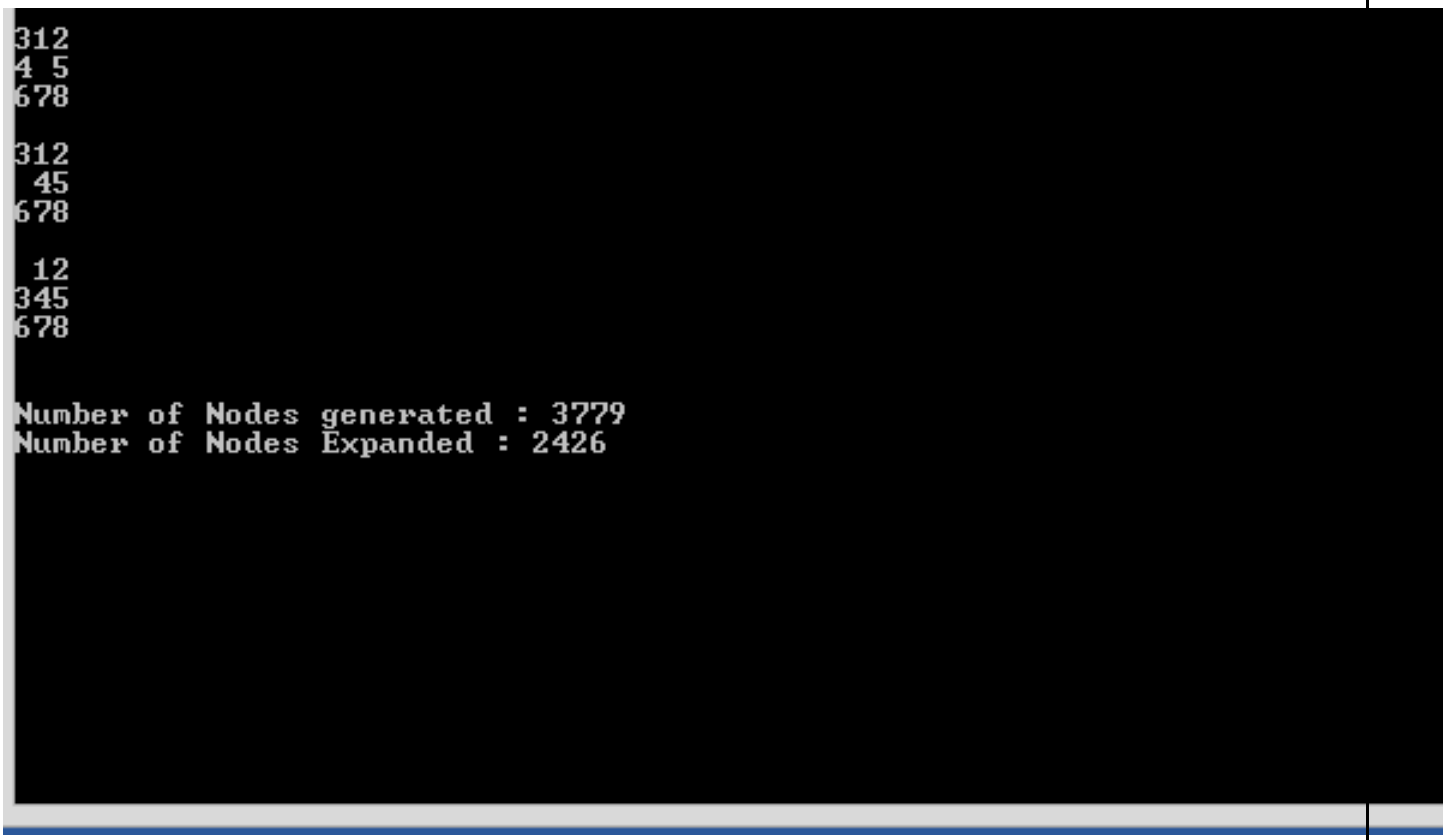

file:///F:/Skydrive/Code/AlgoImpl/AlgoImpl/bin/Debug/AlgoImpl.EXE
Enter the start state:
5
4
0
6
1
8
7
3
2
Enter the goal state:
1
2
3
4
0
5
6
7
8
Number of Nodes in solution path : 22
54
618
732
5 4
618
732
 54
618
732
654
 18
732
654
1 8
732
654
138
7 2
654
138
72
654
13
728
654
1 3
728


file:///F:/Skydrive/Code/AlgoImpl/AlgoImpl/bin/Debug/AlgoImpl.EXE
6 4
153
728
64
153
728
643
15
728
643
1 5
728
643
15
728
43
615
728
4 3
615
728
413
6 5
728
413
625
7 8
413
625
78
413
25
678
13
425
678
1 3
425
678
123
4 5
678
Number of Nodes generated : 393
Number of Nodes Expanded : 234

Example III


file:///F:/Skydrive/Code/AlgoImpl/AlgoImpl/bin/Debug/AlgoImpl.EXE
Enter the start state:
7
2
4
5
0
6
8
3
1
Enter the goal state:
0
1
2
3
4
5
6
7
8
Number of Nodes in solution path : 26
724
5 6
831
724
 56
831
 24
756
831
2 4
756
831
254
7 6
831
254
736
8 1
254
736
 81
254
 36
781
254
3 6
781


file:///F:/Skydrive/Code/AlgoImpl/AlgoImpl/bin/Debug/AlgoImpl.EXE
254
36
781
25
364
781
2 5
364
781
25
364
781
325
64
781
325
6 4
781
325
64
781
325
641
78
325
641
7 8
325
641
78
325
41
678
325
4 1
678
325
41
678
32
415
678


312
4 5
678
312
45
678
12
345
678
Number of Nodes generated : 3779
Number of Nodes Expanded : 2426