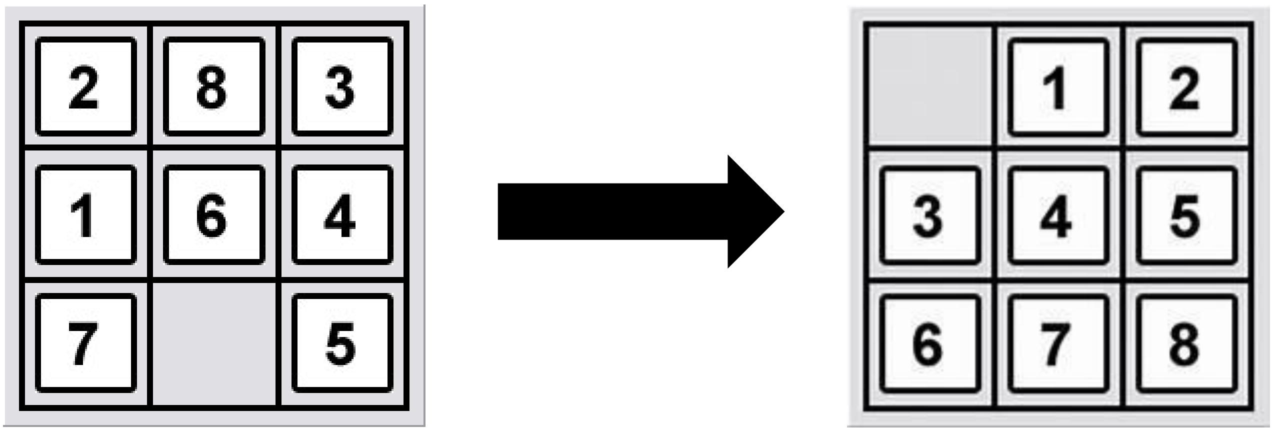
8-PUZZLE SOLVER

الاسم	الرقم الجامعي
عبدالرحمن أحمد يسري النعناني	19015889
أحمد هشام عبدالرزاق	19015378
محمود جاد ابوالوفا	19016532
صلاح الدين أحمد محمد السيد الطنيحي	19015854

8-Puzzle solver

Problem Statement:

8-Puzzle game consists of a board holding 8 distinct movable tiles, plus an empty space. For any such board, the empty space may be legally swapped with any tile horizontally or vertically adjacent to it. Given an initial state of the board, it's required to use AI algorithms to find a sequence of moves that transitions this state to the goal state.



Implemented Algorithms:

This program depends on 3 algorithms:

1- BFS
2- DFS
3- A*: Using 2 different heuristic functions (Manhattan distance, Euclidean distance)

Used Data Structure:

- **Stack**: used in **DFS** algorithm as a frontier
- **Queue**: used in **BFS** algorithm as a frontier
- **Priority Queue (Heap):** used in **A*** algorithm as a frontier
- **List**: used to save any details (expanded states, path to goal)
- **Dictionary**: to save each state to its parent in the tree
- **Set**: to save explored states

Program features

- This program is designed to solve any shape of the puzzle that will be entered (in case It has a solution)
- The program uses 3 algorithms (BFS, DFS, A*)
- The Program can find different information during finding the solution:

- cost of the path from initial state to final state
- path from initial state to final state
- run time to find the solution
- all expanded states
- depth of the tree

Implementation (pseudo code):

```
Read initial state

Check that input is valid

If not valid: read initial state again

Else:

    Choose the algorithm:

    if algorithm is BFS: search using BFS ()

    If algorithm is DFS: search using DFS ()

    If algorithm is A*: search using A*()
```

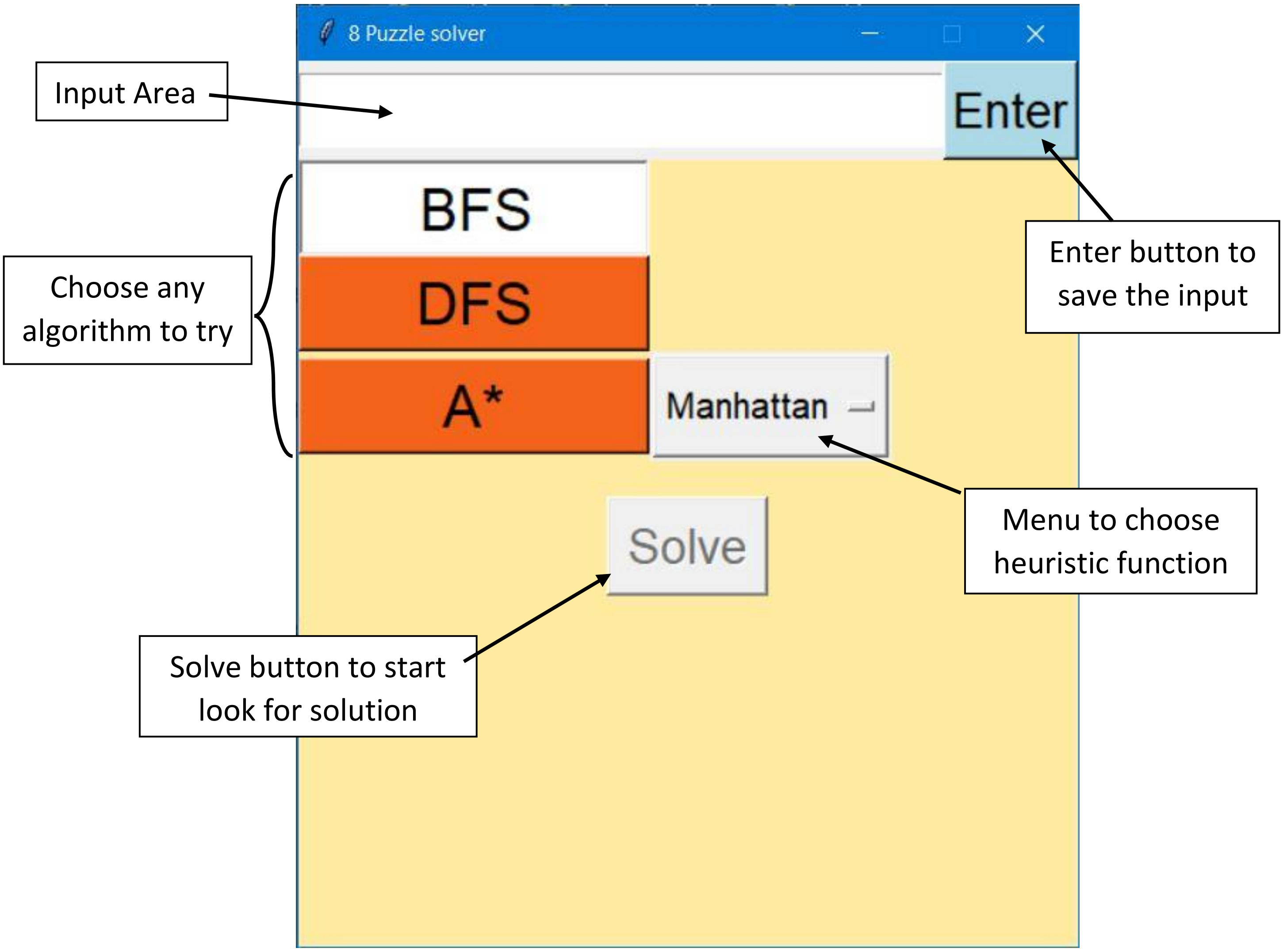
```
BFS (initialState):
    Frontier = Queue.new(initialState)
    Explored = Set.new()
    Parents = hashmap.new(initialState, initialState)
    While not frontier.isEmpty():
        State = fronter.dequeue()
        Explored.add(State)
        If State == goalState:
            Find cost, depth, run-time
            find path goal state to initial state with backtrace
            Return SUCCESS(State)
        For neighbour in state.neighbors():
            If neighbour not in Frontier U Explored:
                Parent[neighbour]  = State
                Frontier.enqueue(neighbour)
        Return FAILURE
```

```
DFS (initialState):
    Frontier = Stack.new(initialState)
    Explored = Set.new()
    Parents = hashmap.new(initialState, initialState)

    While not frontier.isEmpty():
        State = fronter.pop()
        Explored.add(State)
        If State == goalState:
            Find cost, depth, run-time
            find path goal state to initial state with backtrace
            Return SUCCESS(State)

        For neighbour in state.neighbors():
            If neighbour not in Frontier U Explored:
                Parent[neighbour]  = State
                Frontier.push(neighbour)
        Return FAILURE
    --------------------------------------------------------------------------------
```

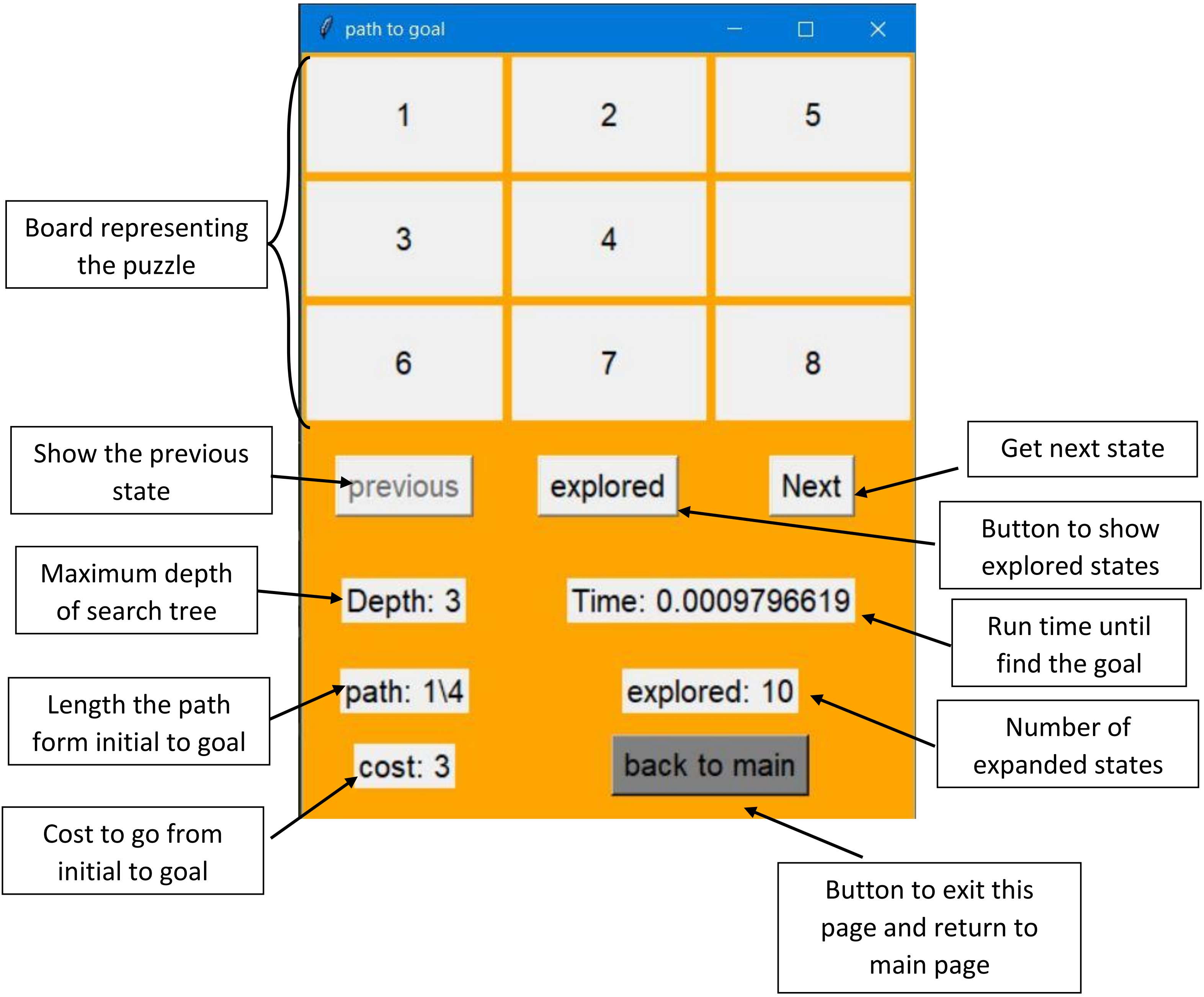
```
A* (initialState):
    Frontier = PriorityQueue.new(initialState)
    Explored = Set.new()
    Parents = hashmap.new(initialState, initialState)
    Frontier.add(initialState)
    While not frontier.isEmpty():
        State = fronter.deleteMin()
        Explored.add(State)
        If State == goalState:
            Return SUCCESS
        For neighbour in state.neighbors():
            If (neighbour not in Frontier U Explored)
                              or (neighbour in frontier) :
                neighbour.depth = state.depth + 1
                calculate_cost(neighbour, neighbour.depth)
                Parent[neighbour]  = State
                Frontier.add(neighbour)
        Return FAILURE
```

How to use the program:

This program uses simple UI:



1- enter the initial state in the input area
2- press enter to save initial state
3- choose any algorithm to use (in case of A* choose the heuristic function)
4- start solving by click solve



5- a new window will open showing the path to the goal, with other information:
   a. number of states from initial state to the goal
   b. number of expanded states
   c. depth of the tree
   d. cost of the path
   e. run time

6- we can go through the path state by state by clicking on next or previous

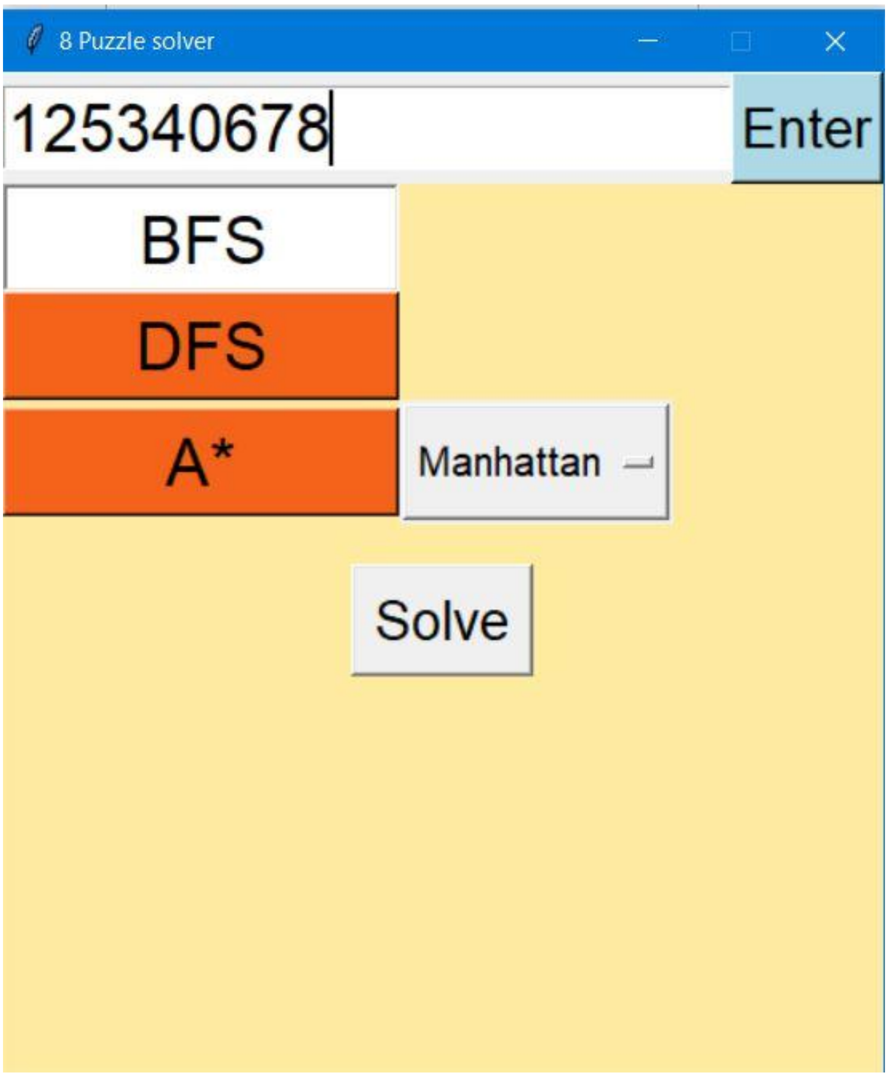
7- on clicking explored a new page will open to see all expanded states
8- to close any pop-up page, click back to main

Assumptions

- the program may take many seconds (can reach 100 sec) to find the solution in some cases

Test cases

1- simple test case
  - Simple input

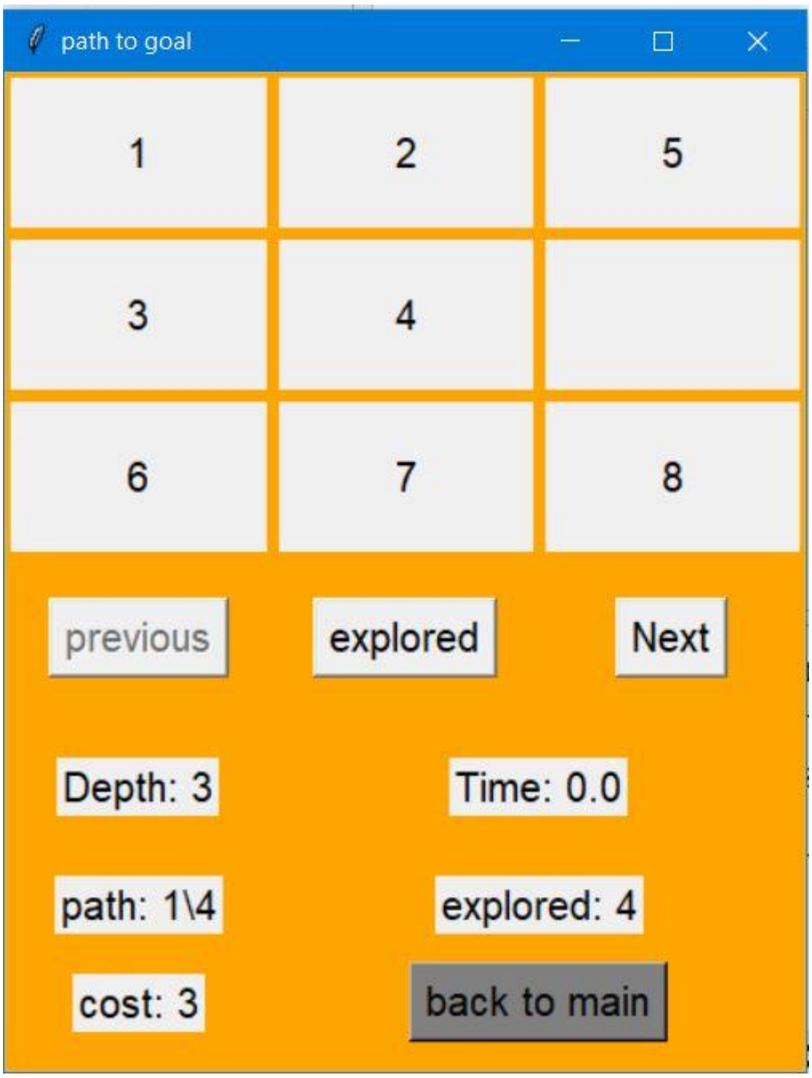


  - Simple A* Manhattan

- Simple A* Euclidian


path to goal
1
2
5
3
4
6
7
8
previous
explored
Next
Depth: 3
Time: 0.000289
path: 1\4
explored: 4
cost: 3
back to main

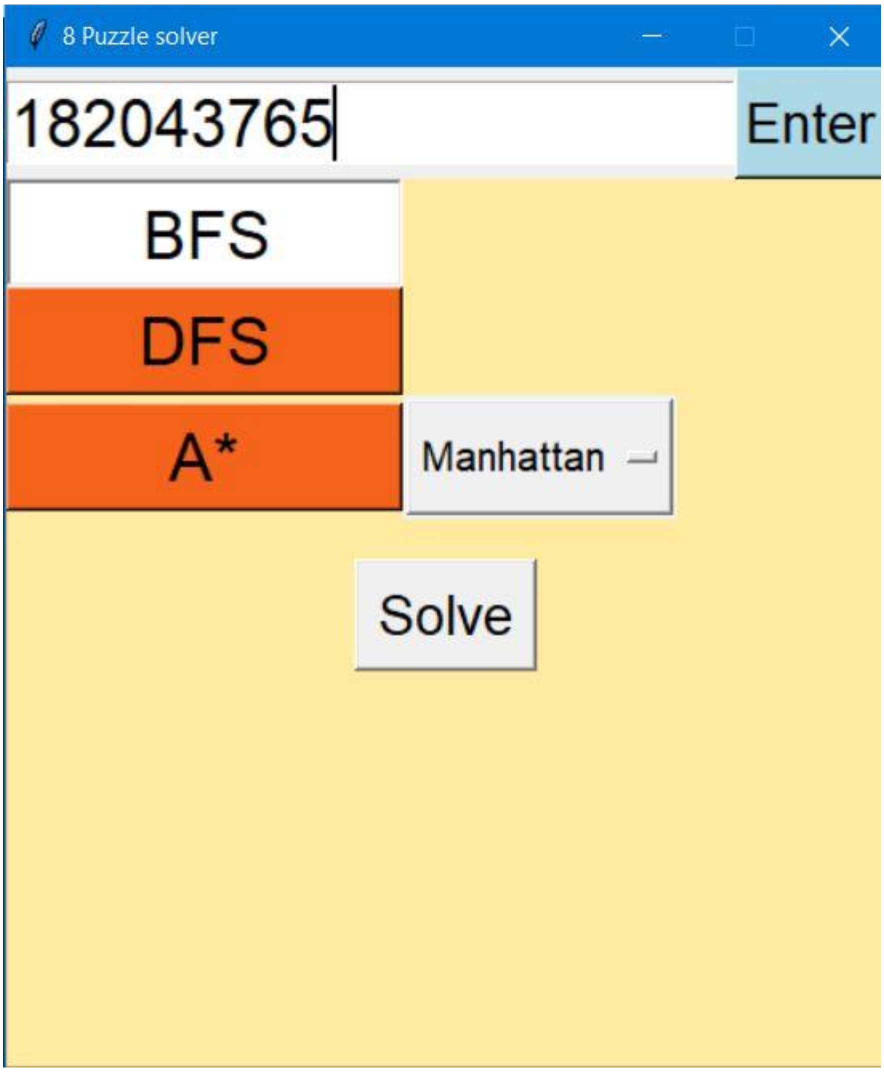
- Simple DFS


path to goal
1
2
5
3
4
6
7
8
previous
explored
Next
Depth: 3
Time: 0.0
path: 1\4
explored: 4
cost: 3
back to main

- Simple BFS

2- hard test case

- Hard input



- Hard A* Manhattan

- Hard A* Eculidian

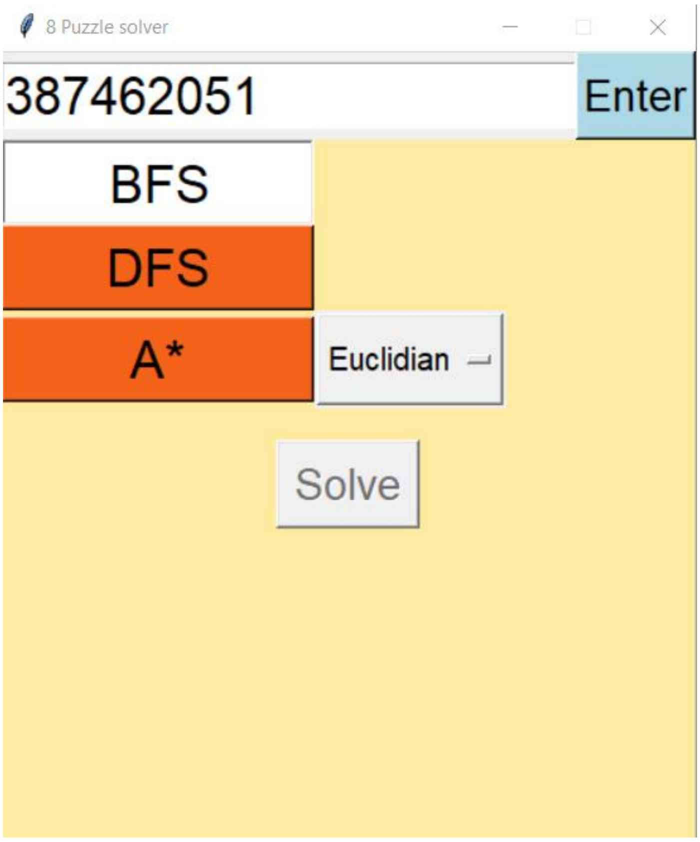

path to goal
1
8
2
4
3
7
6
5
previous
explored
Next
Depth: 21
Time: 0.025167
path: 1\22
explored: 703
cost: 21
back to main

- Hard DFS


path to goal
1
8
2
4
3
7
6
5
previous
explored
Next
Depth: 90655
Time: 84.55030560493469
path: 1\90656
explored: 105582
cost: 90655
back to main

- Hard BFS

Hard teset case 2 :-


8 Puzzle solver
387462051
Enter
BFS
DFS
A*
Euclidian
Solve

BFS :-


path to goal
1
2
3
4
5
6
7
8
previous
explored
Next
Depth: 24
Time: 296.4863033295
path: 25\25
explored: 125044
cost: 24
back to main

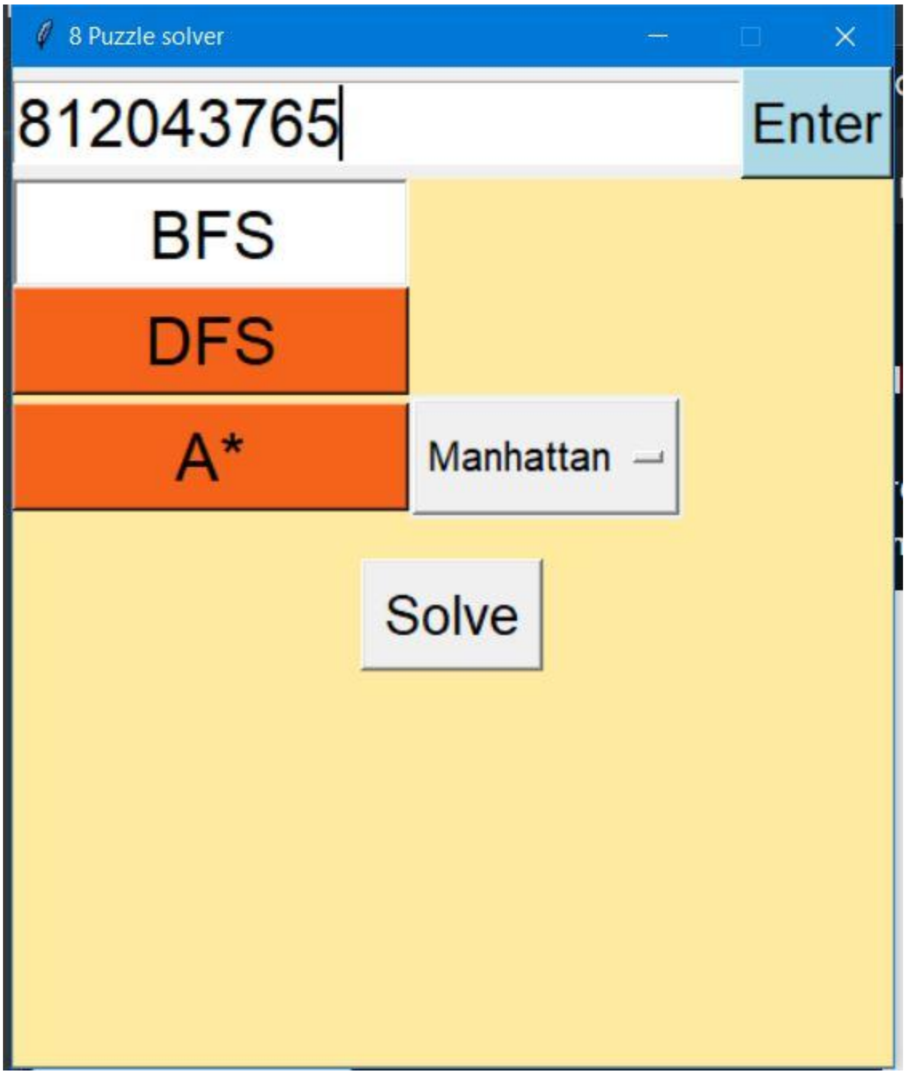
DFS :-

A* Manhattan :-

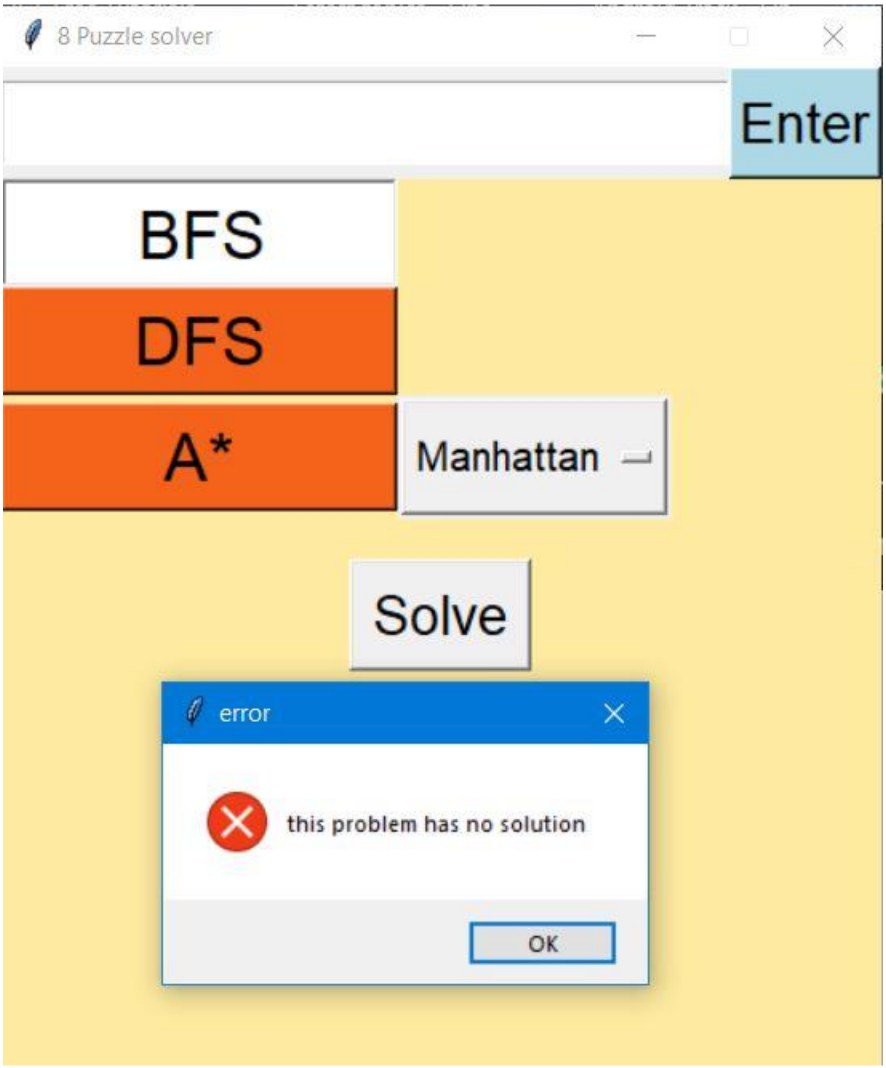
A* Euclidian :-

3- no solution test case

- No solution input

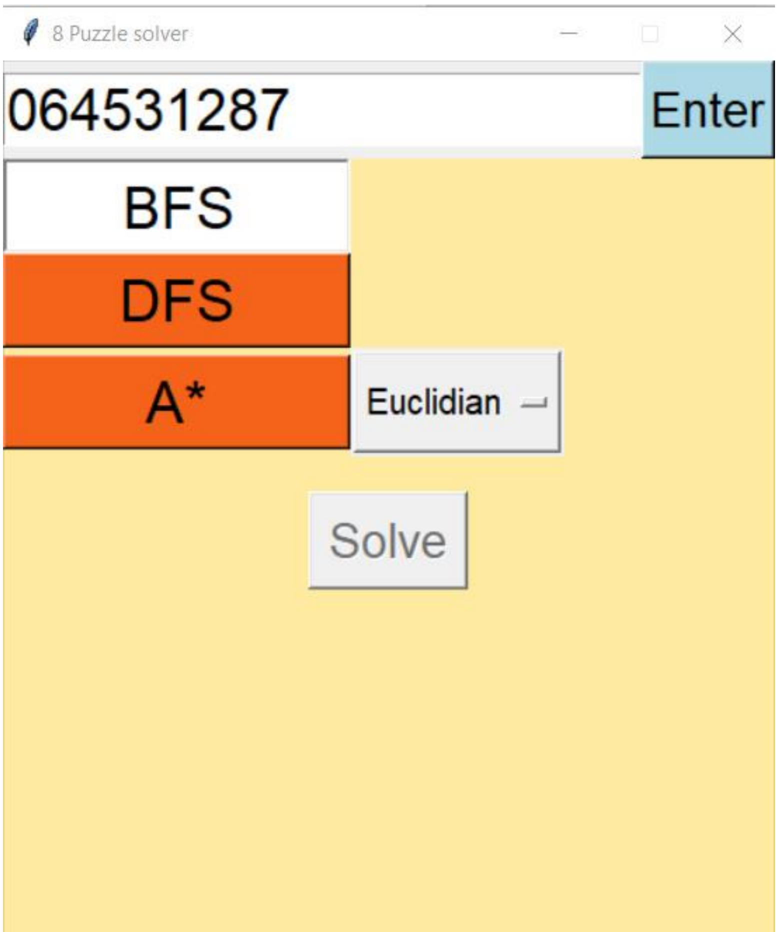


- No solution output



Corner and hard Test case :-

Zero at start :-



BFS :-

DFS :- (FASTER than BFS!)

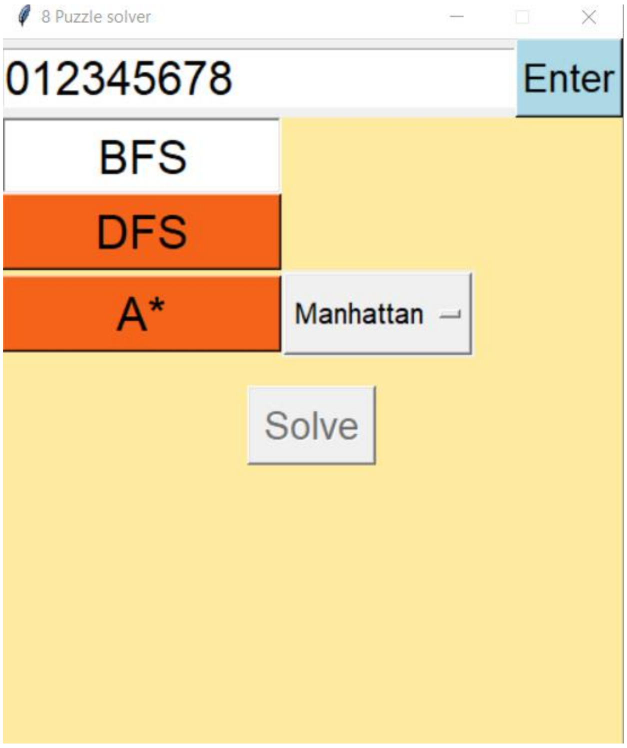

path to goal
6
4
5
3
1
2
8
7
previous
explored
Next
Depth: 40034
Time: 12.24234652519226
path: 1\40035
explored: 41107
cost: 40034
back to main

A* Manhattan :-

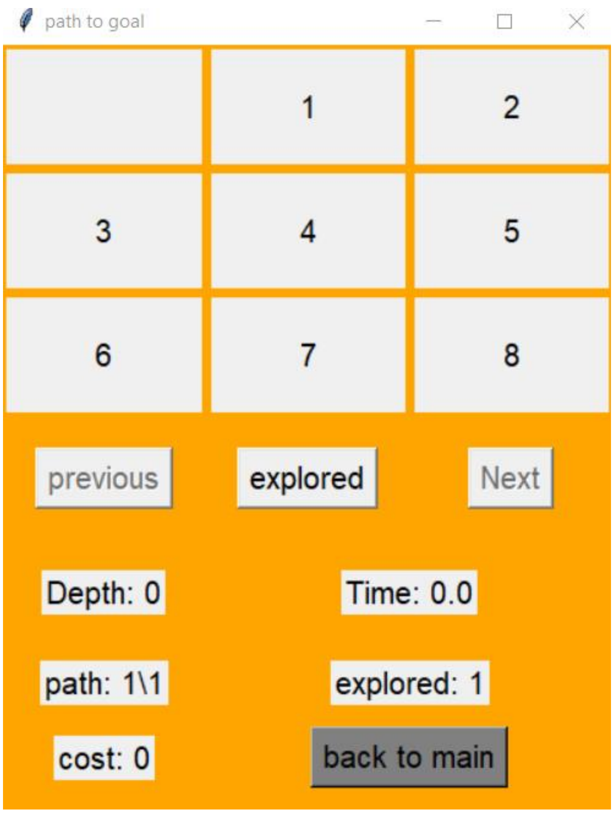

path to goal
6
4
5
3
1
2
8
7
previous
explored
Next
Depth: 26
Time: 0.147109
path: 1\27
explored: 3977
cost: 26
back to main

A* Euclidian :-

Presolved test case :-



BFS :-


path to goal
1
2
3
4
5
6
7
8
previous
explored
Next
Depth: 0
Time: 0.0
path: 1\1
explored: 1
cost: 0
back to main

DFS :-

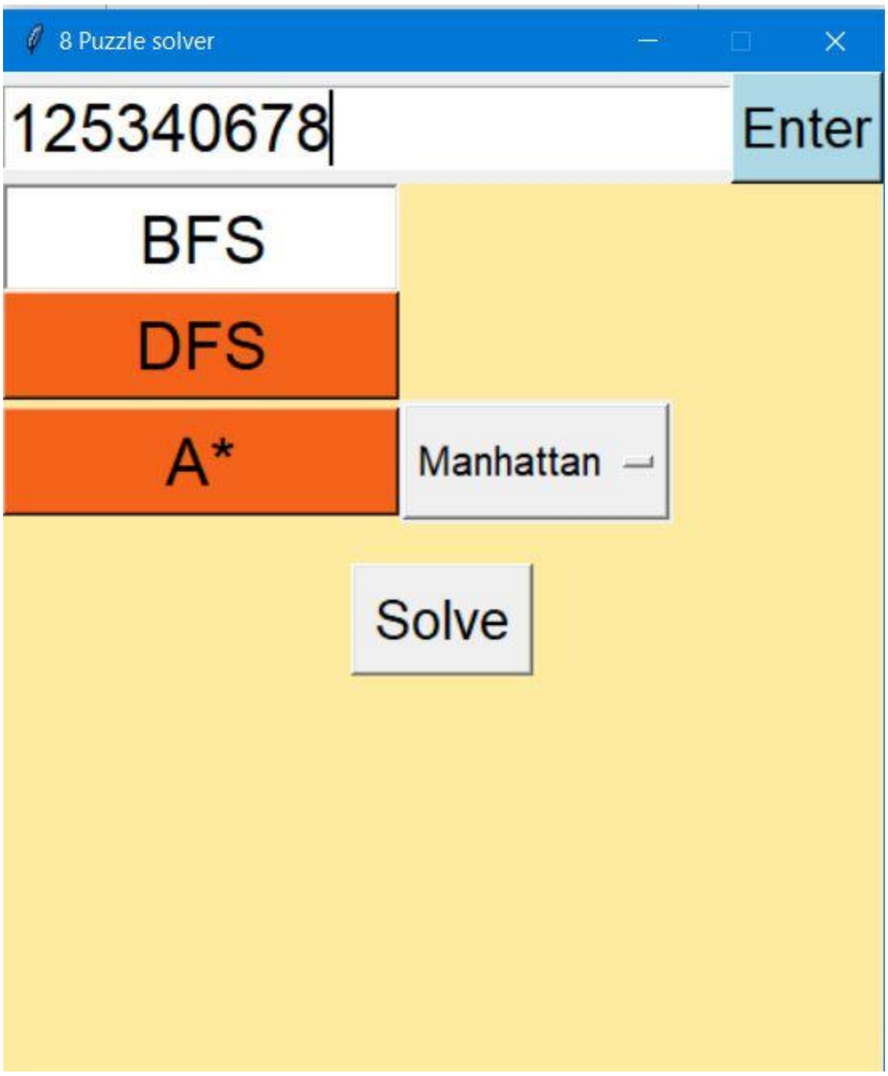

path to goal
1
2
3
4
5
6
7
8
previous
explored
Next
Depth: 0
Time: 0.0
path: 1\1
explored: 1
cost: 0
back to main

A* Manhattan :-

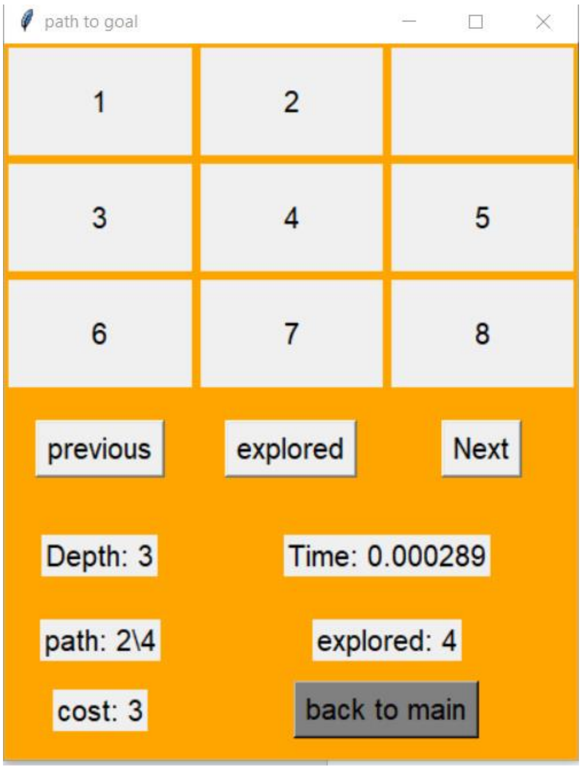

path to goal
1
2
3
4
5
6
7
8
previous
explored
Next
Depth: 0
Time: 7.1e-05
path: 1\1
explored: 1
cost: 0
back to main

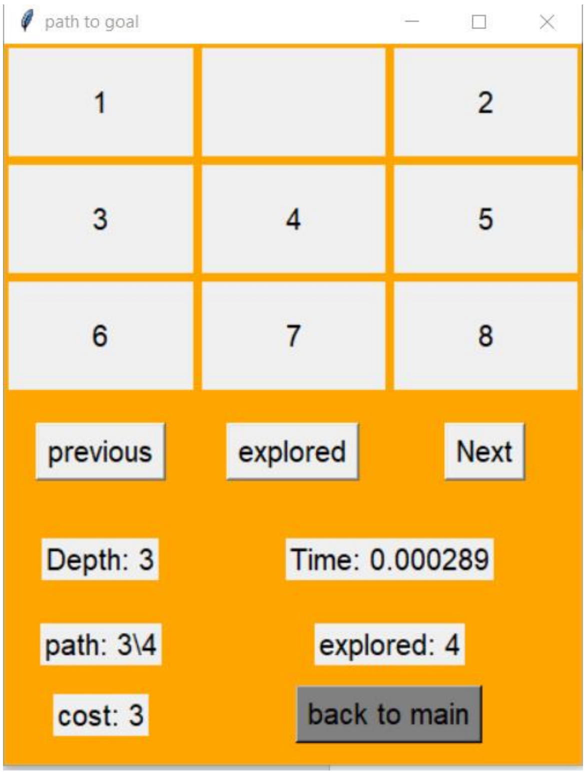
A* Euclidian :-

Example for expanded node view :-


8 Puzzle solver
125340678
Enter
BFS
DFS
A*
Manhattan
Solve


path to goal
1
2
5
3
4
6
7
8
previous
explored
Next
Depth: 3
Time: 0.000198
path: 1\4
explored: 4
cost: 3
back to main


path to goal
1
2
3
4
5
6
7
8
previous
explored
Next
Depth: 3
Time: 0.000289
path: 2\4
explored: 4
cost: 3
back to main


path to goal
1
2
3
4
5
6
7
8
previous
explored
Next
Depth: 3
Time: 0.000289
path: 3\4
explored: 4
cost: 3
back to main


path to goal
1
2
3
4
5
6
7
8
previous
explored
Next
Depth: 3
Time: 0.000289
path: 4\4
explored: 4
cost: 3
back to main