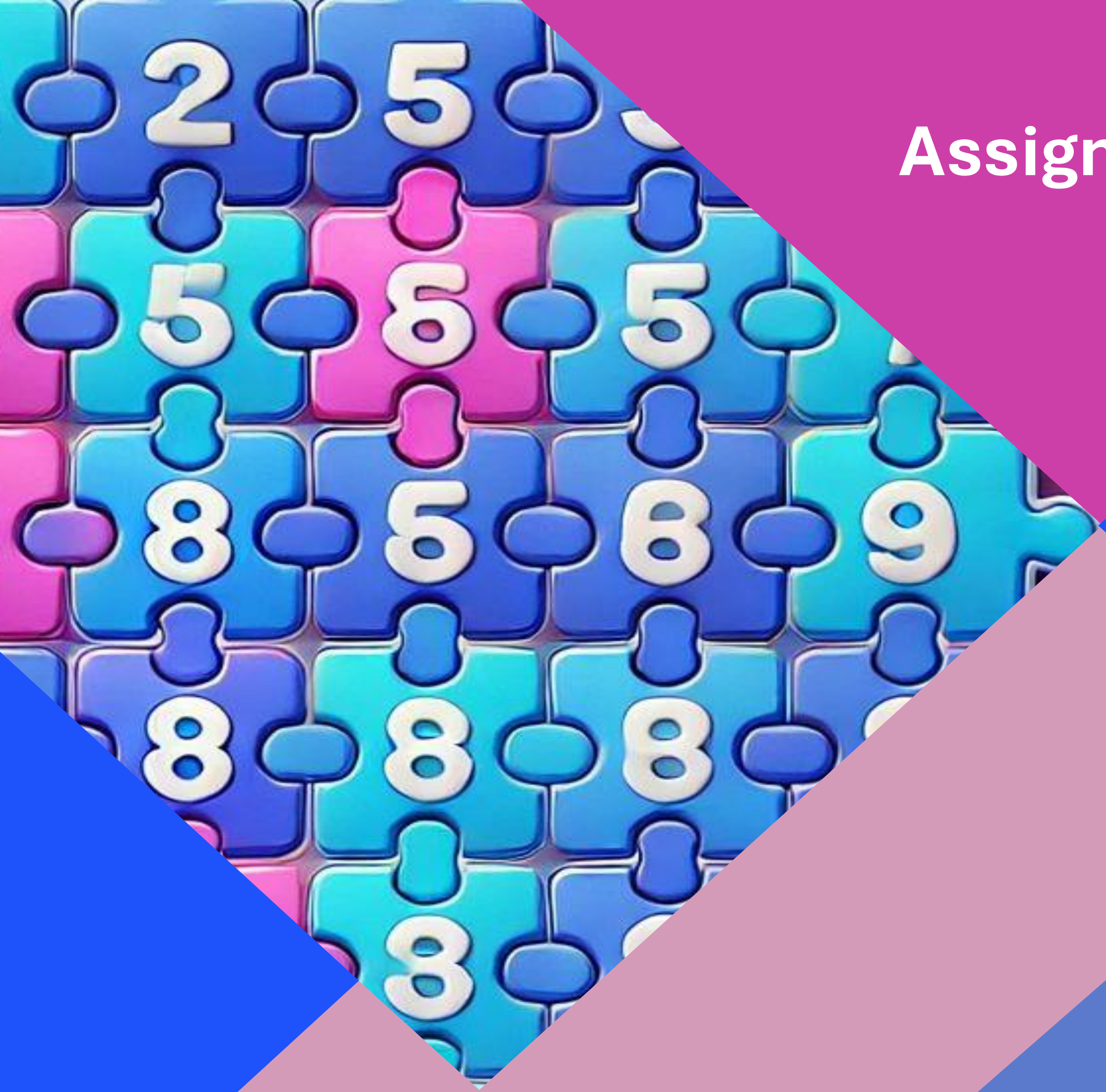
Assignment (1)



# 8-PUZZLE SOLVER USING SEARCH ALGORITHMS

# 8-Puzzle Report

## Team members:

| | |
|---|---|
| Omnia Karem Mahmoud Mahfouz | 21010299 |
| Nouran Ashraf Yusef | 21011492 |
| Rana Mohamed Ali Attia | 21010528 |

## Algorithms:

### DFS:

DFS is a search algorithm that explores as far as possible along one branch before backtracking

- **Data Structures:**
  1- Stack (via list): frontier.
  2- List: path and moves.
  3- Set: explored.
  4- Dictionary: parents and direction.
  5- Tuple: for frontier elements.

- **Algorithm Explanation:**
  1- Frontier is initialized with the start state and depth 0.
  2- If the state is already goal_state, the solution is returned immediately.
  3- Each state popped from the frontier is marked as explored.
  4- The children of the current state are generated by moving the empty tile (0) in the possible directions (left, right, up, down).
  5- For each child, if it's not in the explored set or frontier, it is added to the frontier along with the current depth.
  6- If the goal state is reached, the path and moves are reconstructed using the parents and direction dictionaries.
  7- If the frontier becomes empty and the goal is not found, it returns None (indicating no solution).

## IDS:

IDS is a search strategy that combines the benefits of DFS and BFS. It performs a series of DFS searches, incrementally increasing the depth limit at each iteration, until the goal is found.

- **Data Structures:**
    1- Stack (via list): frontier.
    2- List: path and moves.
    3- Set: explored.
    4- Dictionary: parents and direction.
    5- Tuple: for frontier elements.

- **Algorithm Explanation:**
    1- For each depth limit from 0 to max_depth, it calls Depth-Limited Search (DLS).
    2- If DLS finds a solution, IDS stops and returns the path.
    3- If no solution is found, IDS increases the depth and tries again.
    4- DLS Uses a stack-based approach to explore states.
    5- Stores parent-child relationships
    using parents and direction dictionaries for backtracking.
    6- Expands only if the current depth is less than the depth limit.

## BFS:

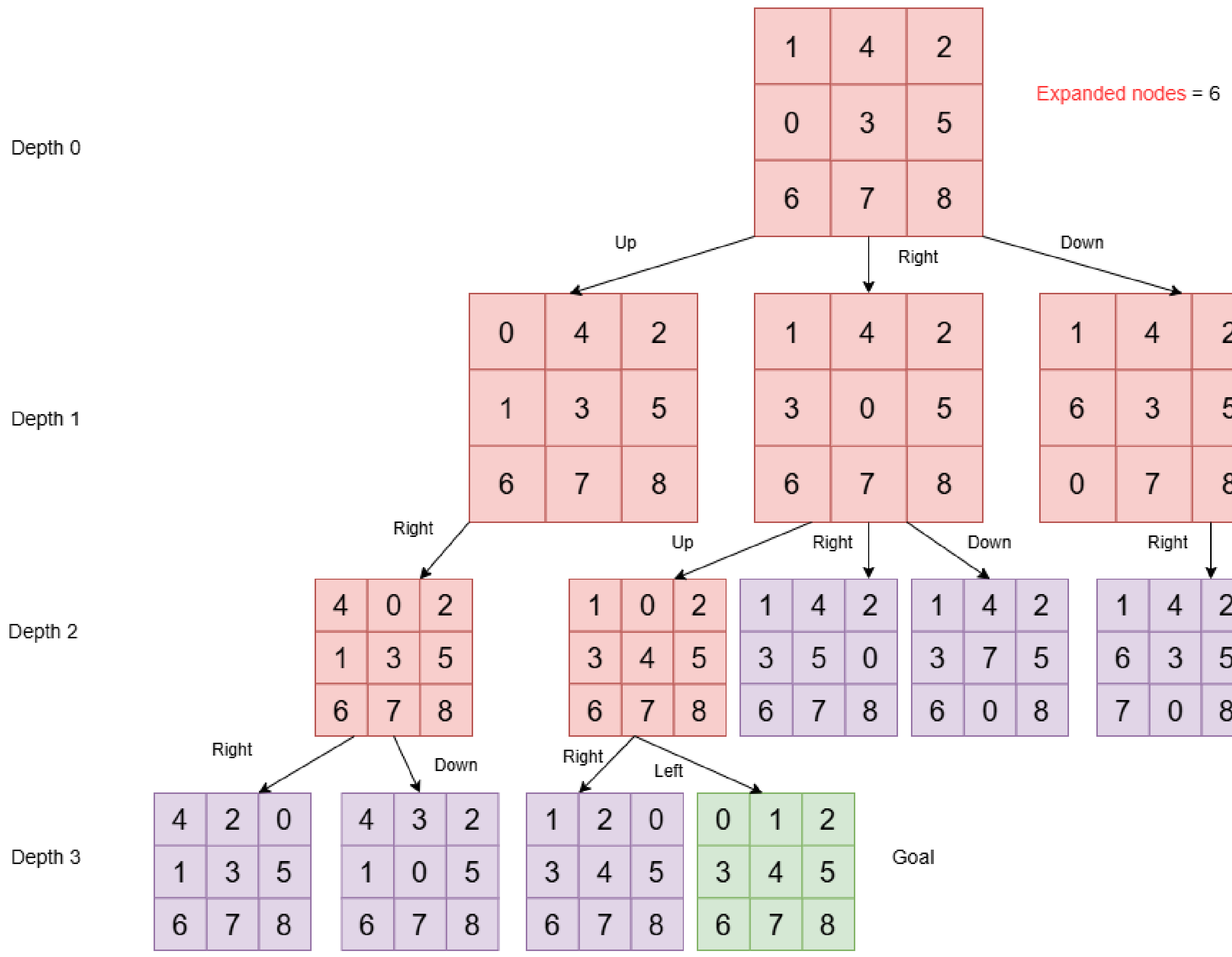
Breath first search algorithm explores all the nodes at a level before going deep exploring other neighbors (explores shallowest first). Hence, it provides the optimal solution and the shortest solution path in the 8-puzzle problem.

- **Data structures used:**
    1. **Queue:** which represents the frontier, which stores the states in FIFO manner along with the depth of that node.
    2. **Set(visited):** to keep track of visited states.
    3. **Dictionary(parents):** used to store the parents of each state and the action that lead to it which will be used in constructing the path.
    4. **Tuple:** to hold the elements in the frontier, containing the current board state and depth level.

- ## Algorithm Explanation:

  1. **Initial Checks:**
     Check if the initial board state is **already solved** or it is **not solvable** and terminate if so.
  2. **Initialization:**
     - Create a queue (frontier) to keep track of states to explore, initializing it with the initial board state and depth level of 1.
     - Maintain a set (visited) to record states that have already been explored, starting with the initial state.
     - Initialize counters for nodes_expanded and max_depth to track the number of nodes explored and the maximum depth reached.
     - Use a dictionary (parents) to store the parent of each state and the action leading to it.
     - Record the start time to calculate the total execution time later.
  3. **Main Loop:**
  **While there are states in the frontier:**
  - **Pop State**: remove the first state from the queue and mark it as visited.
  - Increment the nodes_expanded counter.
  - Update max_depth if the current depth exceeds the previous maximum.
  - **Explore Neighbors**: For each neighbor:
    - Check if the neighbor has not been visited and is not already in the frontier.
    - If so, add this neighbor to the frontier with an incremented depth.
    - Record the parent state and action leading to this neighbor.
    - **Check for Goal State**: If the neighbor matches the target state:
      - Record the end time.
      - Construct the solution path.
      - Calculate the cost (number of moves to reach the goal).
      - Return the series of actions taken, cost, number of nodes expanded, maximum depth reached, and execution time.
  4. **Termination (Frontier is empty):**
     If all nodes are explored and the target state is not found, return "No Solution".

- ## How the algorithm operates:
- The initial state is shown at depth 0, where the board state is 142035678.
- The algorithm starts by exploring the neighboring states at depth 1.
- These neighboring states are added to the frontier and while expanding it checks if the neighbor is the goal.
- If the goal is not reached then the algorithm pops the next state from the frontier and mark it as visited to avoid revisiting it then explores its neighbors at depth 2, again adding them to the frontier and checking if goal is reached.
- This process continues, level by level, exploring all possible configurations until the goal state (01234567) is found.



## A* Algorithm:

A* algorithm is an informed searching algorithm, which means that it takes decision based on something, in A* we use a heuristic to decide which path to take next. Heuristic helps us to take steps closer to the goal than exploring further states that won't be useful to us.

| Attribute | A* Manhattan Distance | A* Euclidean Distance |
|---|---|---|
| **Data Structure** | • **Set**: Used in `explored` to check if a child node has been explored, with an average time complexity of O(1).<br>• **Dictionary (`parents`)**: Stores each state's parent and the action leading to it, useful for constructing the path.<br>• **Priority Queue (`heapq`)**: Retrieves the state with the least cost, ensuring nodes closer to the goal are explored before those further away.<br>• **Tuple**: Holds the elements in the frontier, containing the cost, depth, and current state. | • **Set**: Used in `explored` to check if a child node has been explored, with an average time complexity of O(1).<br>• **Dictionary (`parents`)**: Stores each state's parent and the action leading to it, useful for constructing the path.<br>• **Priority Queue (`heapq`)**: Retrieves the state with the least cost, ensuring nodes closer to the goal are explored before those further away.<br>• **Tuple**: Holds the elements in the frontier, containing the cost, depth, and current state. |

| | | |
|---|---|---|
| **Calculation** | h = abs(current cell:x - goal:x) + abs(current cell:y - goal:y)<br>g = depth<br>f = g + h | sqrt((current cell:x - goal:x)2 + (current cell.y - goal:y)2<br>)<br>g = depth<br>f = g + h |
| **Admissible** | The **Manhattan heuristic** is more admissible for grid-based problems like the 8-puzzle, where it accurately estimates the number of moves to reach the goal since moves are restricted to vertical and horizontal directions. | The **Euclidean heuristic** is admissible but less effective than Manhattan in grid-based problems. Although it represents the straight-line (diagonal) distance, it doesn't align as well with the movement constraints in the 8-puzzle. It's better suited to map-based problems with diagonal movements. |
| **Explanation** | In A*, the algorithm calculates the cost and prioritizes states that are closer to the goal. It achieves this by summing the depth **g** with the heuristic **h** to form the cost **f**.<br><br>With Manhattan, the heuristic represents the sum of the moves required for each piece to reach its goal state. This prioritization allows A* to explore the state with the lowest cost first. For instance, if two states have costs 2 and 5, the state with cost 2 will be evaluated first, possibly leading to the optimal solution.<br><br>A* guarantees optimality when the heuristic is admissible. | The A* algorithm with Euclidean heuristic works similarly by calculating the total cost and prioritizing states near the goal. Here, the heuristic represents the straight-line distance, which isn't as accurate in grid-based puzzles but can lead to faster goal-reaching than BFS. Although it may not always yield the optimal solution in grid-based constraints, Euclidean is still more efficient for cases where the goal is aligned with diagonal movements. |

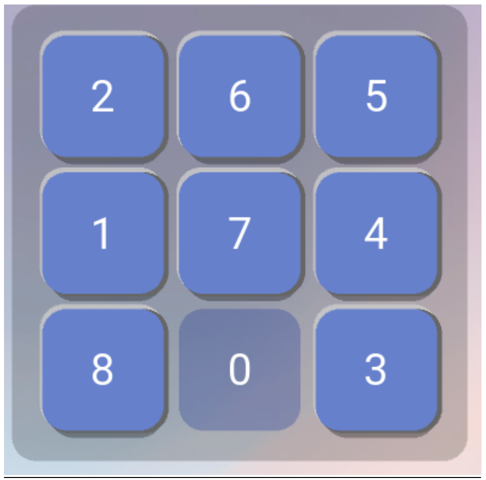
## Testcases:

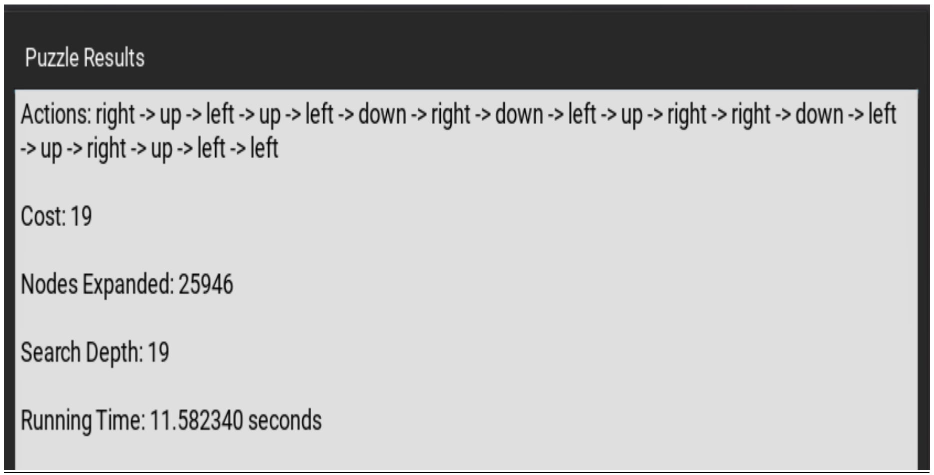
| Testcase | Algorithm | Cost | Nodes Expanded | Search Depth | Running time |
|---|---|---|---|---|---|
| **328451670** | BFS | 12 | 980 | 12 | 0.049 |
| | DFS | 51492 | 53291 | 51491 | 50.316732 |
| | IDS | 12 | 735 | 12 | 0.03 |
| | A*(Euclidean) | 12 | 34 | 12 | 0.003 |
| | A*(Manhattan) | 12 | 34 | 12 | 0.0019 |
| **725310648** | BFS | 15 | 3820 | 15 | 0.32 |
| | DFS | 42111 | 43310 | 42110 | 33.389141 |
| | IDS | 15 | 6810 | 15 | 0.11 |
| | A*(Euclidean) | 15 | 153 | 15 | 0.011 |
| | A*(Manhattan) | 15 | 153 | 15 | 0.01 |
| **035428617** | BFS | 10 | 333 | 10 | 0.009988 |
| | DFS | 9634 | 9716 | 9633 | 1.658994 |
| | IDS | 10 | 457 | 10 | 0.0136 |
| | A*(Euclidean) | 10 | 13 | 10 | 0.001 |
| | A*(Manhattan) | 10 | 13 | 10 | 0.001 |
| **641302758** | BFS | 14 | 3211 | 14 | 0.243 |
| | DFS | 75133 | 80938 | 75133 | 128.192308 |
| | IDS | 14 | 2840 | 14 | 0.0854 |
| | A*(Euclidean) | 14 | 130 | 14 | 0.0086 |
| | A*(Manhattan) | 14 | 130 | 14 | 0.006 |
| **158327064** | BFS | 12 | 1059 | 12 | 0.042 |
| | DFS | 94808 | 120326 | 94816 | 283.933433 |
| | IDS | 12 | 296 | 12 | 0.023 |
| | A*(Euclidean) | 12 | 21 | 12 | 0.002 |
| | A*(Manhattan) | 12 | 21 | 12 | 0.002 |

## Sample runs:

### BFS:

**a.**





Puzzle Results

Actions: right -> up -> left -> up -> left -> down -> right -> down -> left -> up -> right -> right -> down -> left -> up -> right -> up -> left -> left

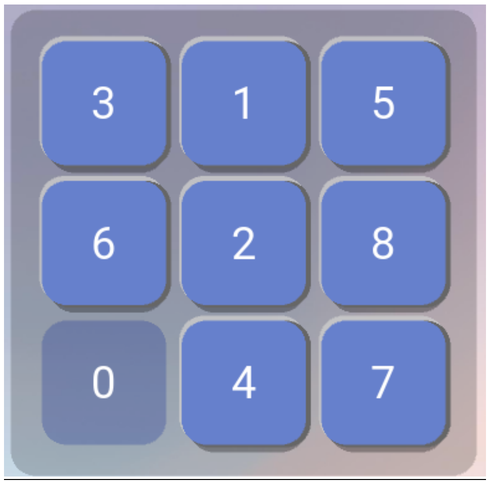
Cost: 19

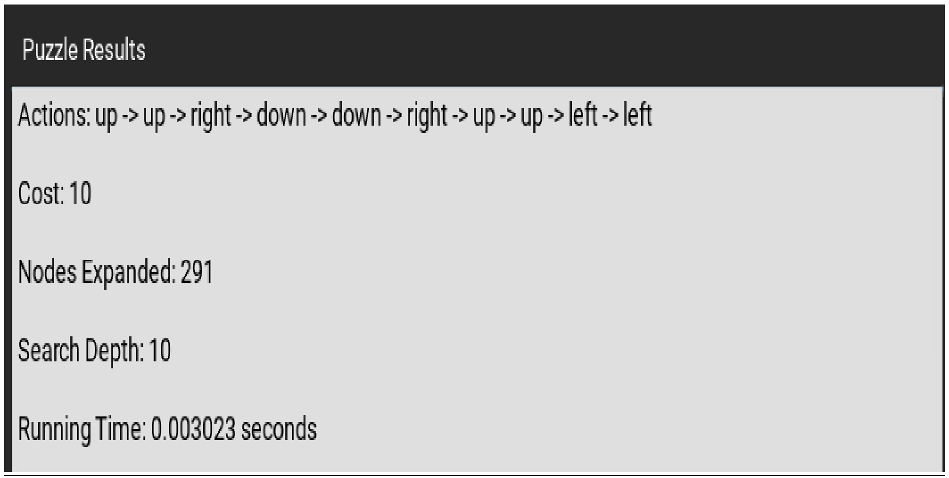
Nodes Expanded: 25946

Search Depth: 19

Running Time: 11.582340 seconds

**b.**





Puzzle Results

Actions: up -> up -> right -> down -> down -> right -> up -> up -> left -> left

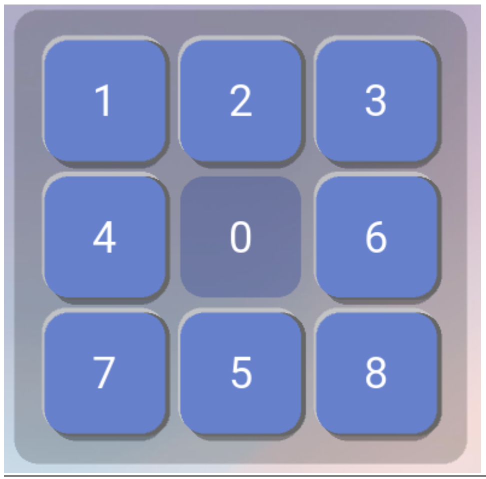
Cost: 10

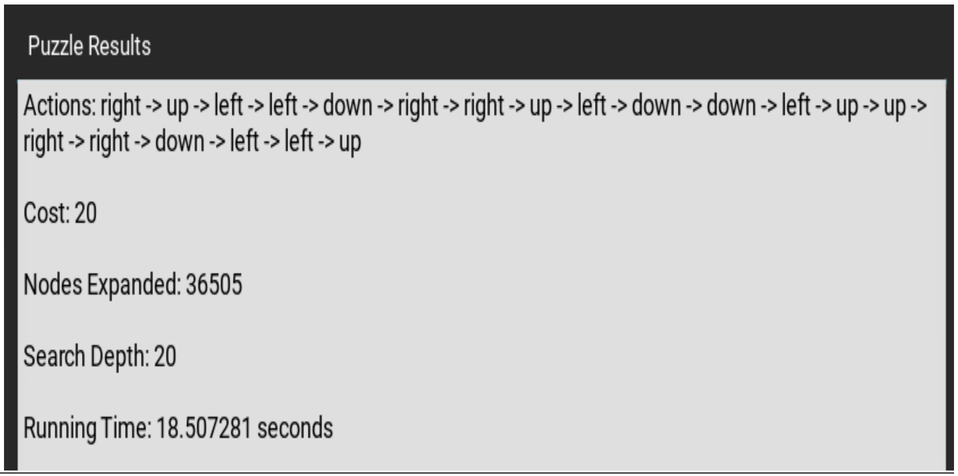
Nodes Expanded: 291

Search Depth: 10

Running Time: 0.003023 seconds

**c.**





Puzzle Results

Actions: right -> up -> left -> left -> down -> right -> right -> up -> left -> down -> down -> left -> up -> up -> right -> right -> down -> left -> left -> up

Cost: 20

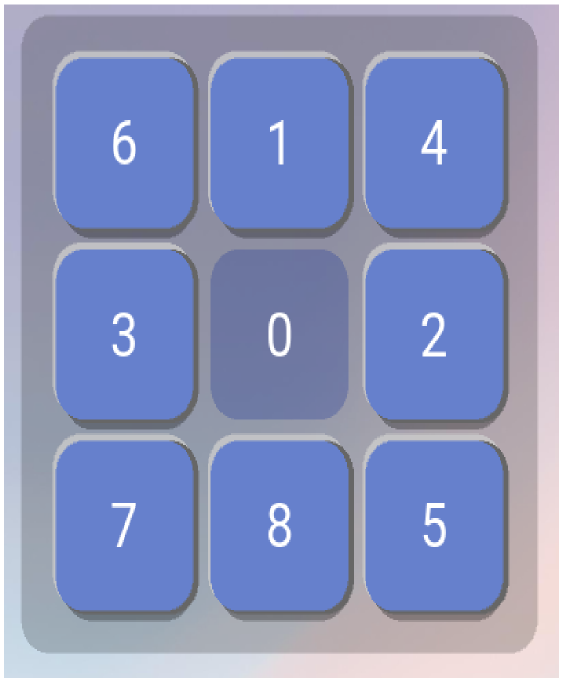
Nodes Expanded: 36505

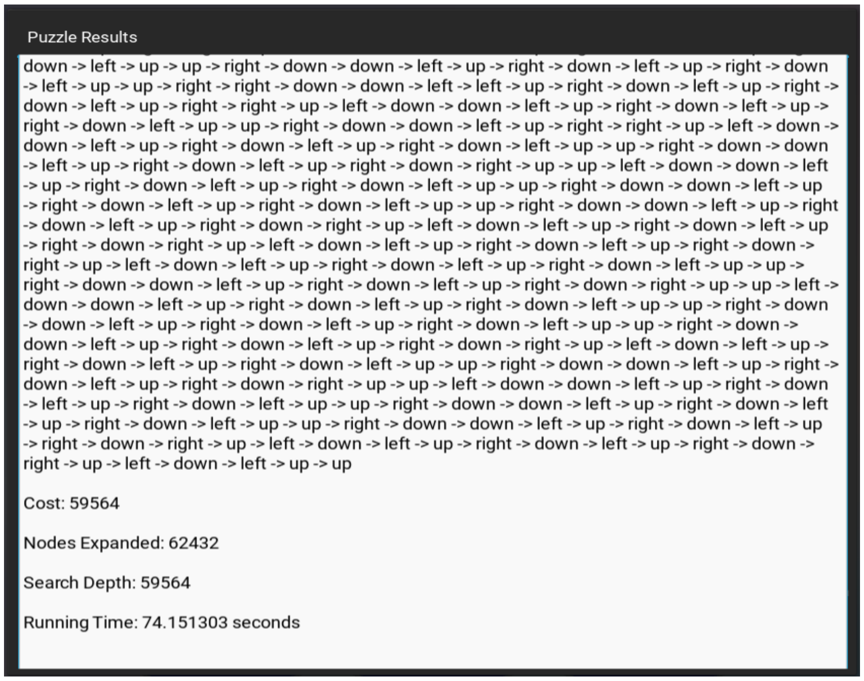
Search Depth: 20

Running Time: 18.507281 seconds

**DFS:** actions are terminated due to its large size.

**a.**

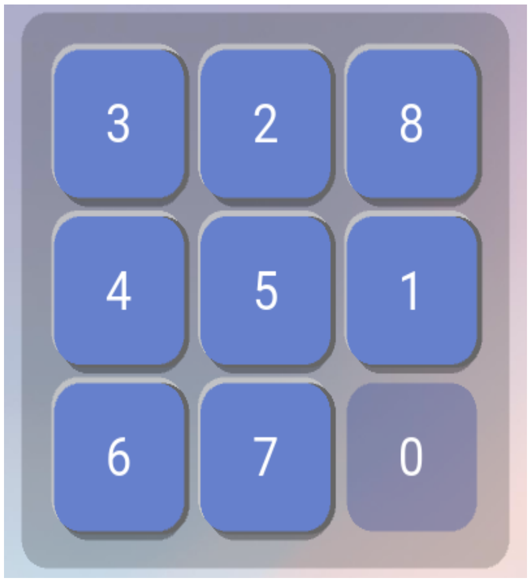


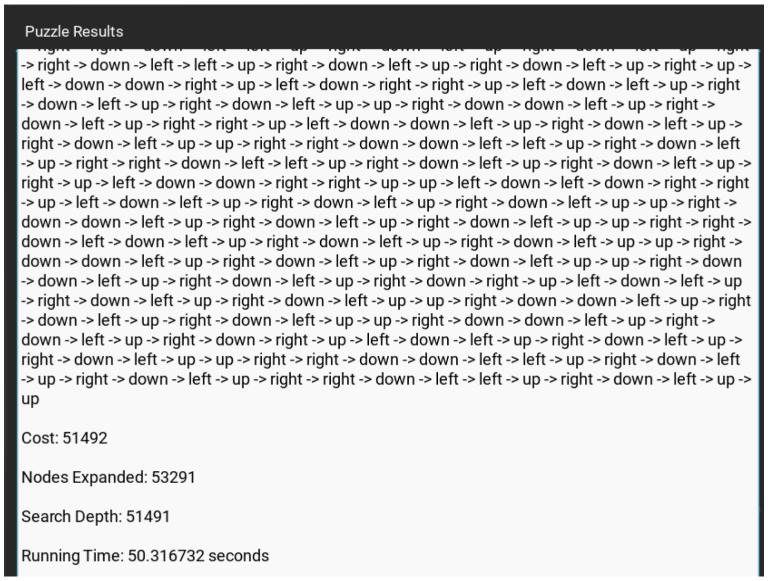


Puzzle Results

```
down -> left -> up -> up -> right -> down -> down -> left -> up -> right -> down -> left -> up -> right -> down
-> left -> up -> up -> right -> right -> down -> down -> left -> left -> up -> right -> down -> left -> up -> right ->
down -> left -> up -> right -> right -> up -> left -> down -> down -> left -> up -> right -> down -> left -> up ->
right -> down -> left -> up -> up -> right -> down -> down -> left -> up -> right -> right -> up -> left -> down ->
down -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> up -> right -> down -> down
-> left -> up -> right -> down -> left -> up -> right -> down -> right -> up -> up -> left -> down -> down -> left
-> up -> right -> down -> left -> up -> right -> down -> left -> up -> up -> right -> down -> down -> left -> up
-> right -> down -> left -> up -> right -> down -> left -> up -> up -> right -> down -> down -> left -> up -> right
-> down -> left -> up -> right -> down -> right -> up -> left -> down -> left -> up -> right -> down -> left -> up
-> right -> down -> right -> up -> left -> down -> left -> up -> right -> down -> left -> up -> right -> down ->
right -> up -> left -> down -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> up ->
right -> down -> down -> left -> up -> right -> down -> left -> up -> right -> down -> right -> up -> up -> left ->
down -> down -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> up -> right -> down
-> down -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> up -> right -> down ->
down -> left -> up -> right -> down -> left -> up -> right -> down -> right -> up -> left -> down -> left -> up ->
right -> down -> left -> up -> right -> down -> left -> up -> up -> right -> down -> down -> left -> up -> right ->
down -> left -> up -> right -> down -> right -> up -> up -> left -> down -> down -> left -> up -> right -> down
-> left -> up -> right -> down -> left -> up -> up -> right -> down -> down -> left -> up -> right -> down -> left
-> up -> right -> down -> left -> up -> up -> right -> down -> down -> left -> up -> right -> down -> left -> up
-> right -> down -> right -> up -> left -> down -> left -> up -> right -> down -> left -> up -> right -> down ->
right -> up -> left -> down -> left -> up -> up

Cost: 59564

Nodes Expanded: 62432

Search Depth: 59564

Running Time: 74.151303 seconds
```

**b.**

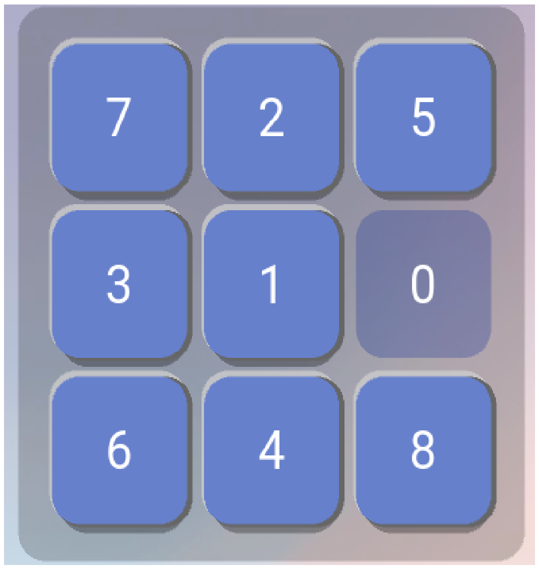


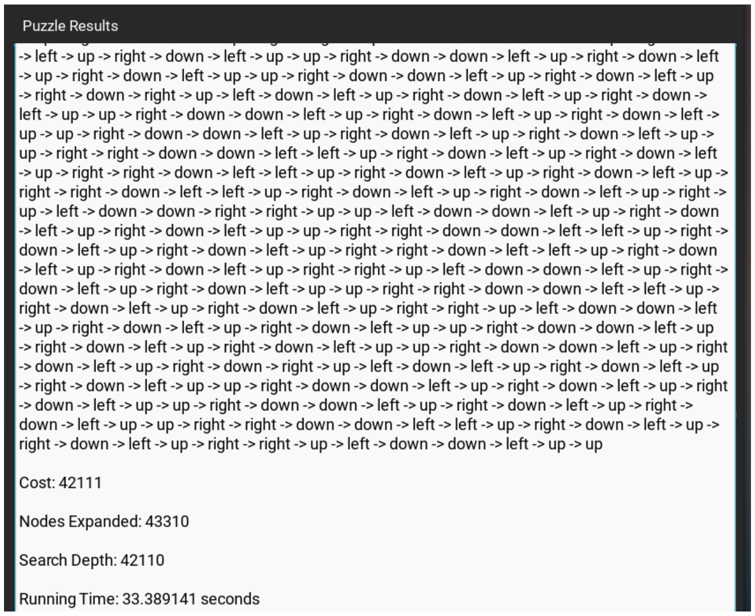


Puzzle Results

```
-> right -> down -> left -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> right -> up ->
left -> down -> down -> right -> up -> left -> down -> right -> right -> up -> left -> down -> left -> up -> right
-> down -> left -> up -> right -> down -> left -> up -> up -> right -> down -> down -> left -> up -> right ->
down -> left -> up -> right -> right -> up -> left -> down -> down -> left -> up -> right -> down -> left -> up ->
right -> down -> left -> up -> up -> right -> right -> down -> down -> left -> left -> up -> right -> down -> left
-> up -> right -> right -> down -> left -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up ->
right -> up -> left -> down -> down -> right -> right -> up -> up -> left -> down -> left -> down -> right -> right
-> up -> left -> down -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> up -> right ->
down -> down -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> up -> right -> right ->
down -> left -> down -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> up -> right ->
down -> down -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> up -> right -> down
-> down -> left -> up -> right -> down -> left -> up -> right -> down -> right -> up -> left -> down -> left -> up
-> right -> down -> left -> up -> right -> down -> left -> up -> up -> right -> down -> down -> left -> up -> right
-> down -> left -> up -> right -> down -> left -> up -> up -> right -> down -> down -> left -> up -> right ->
down -> left -> up -> right -> down -> right -> up -> left -> down -> left -> up -> right -> down -> left -> up ->
right -> down -> left -> up -> up -> right -> right -> down -> down -> left -> left -> up -> right -> down -> left
-> up -> right -> down -> left -> up -> right -> right -> down -> left -> left -> up -> right -> down -> left -> up ->
up

Cost: 51492

Nodes Expanded: 53291

Search Depth: 51491

Running Time: 50.316732 seconds
```

**c.**

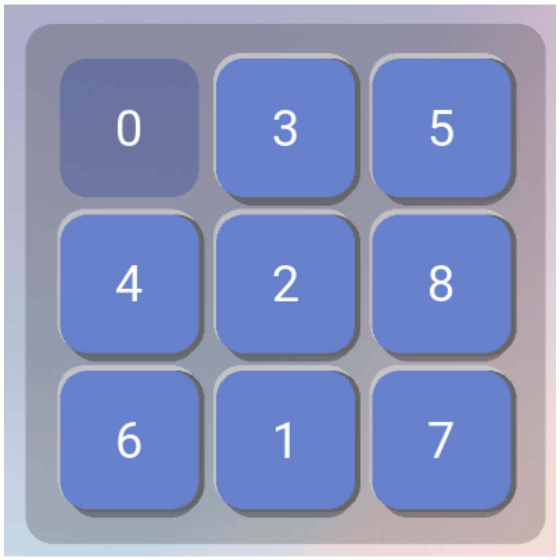


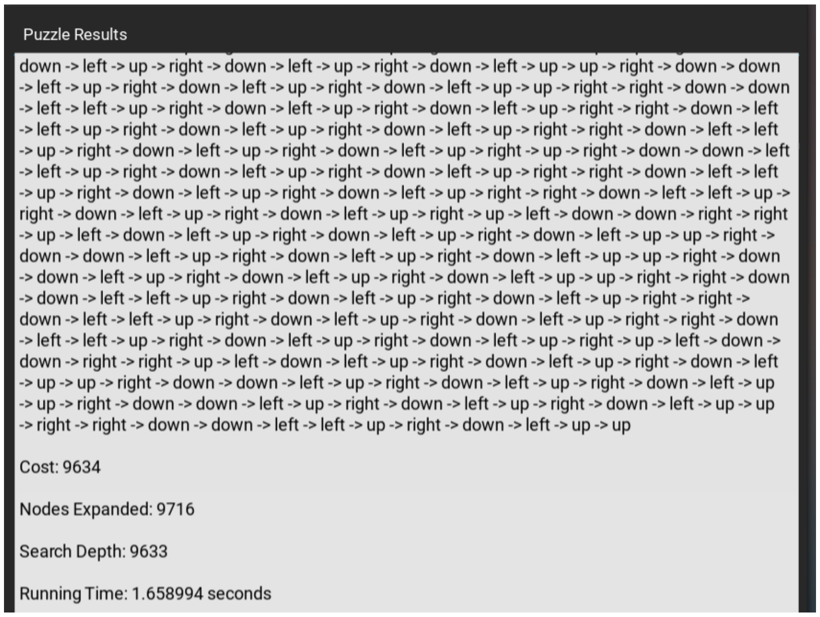


Puzzle Results

```
-> left -> up -> right -> down -> left -> up -> up -> right -> down -> down -> left -> up -> right -> down -> left
-> up -> right -> down -> left -> up -> up -> right -> down -> down -> left -> up -> right -> down -> left -> up
-> right -> down -> right -> up -> left -> down -> left -> up -> right -> down -> left -> up -> right -> down ->
left -> up -> up -> right -> down -> down -> left -> up -> right -> down -> left -> up -> right -> down -> left ->
up -> up -> right -> down -> down -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up ->
up -> right -> right -> down -> down -> left -> left -> up -> right -> down -> left -> up -> right -> down -> left
-> up -> right -> right -> down -> left -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up ->
right -> right -> down -> left -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> right ->
up -> left -> down -> down -> right -> right -> up -> up -> left -> down -> down -> left -> up -> right -> down
-> left -> up -> right -> down -> left -> up -> up -> right -> right -> down -> down -> left -> left -> up -> right ->
down -> left -> up -> right -> down -> left -> up -> right -> right -> down -> left -> left -> up -> right -> down
-> left -> up -> right -> down -> left -> up -> right -> right -> up -> left -> down -> down -> left -> up -> right ->
down -> left -> up -> right -> down -> left -> up -> up -> right -> right -> down -> down -> left -> left -> up ->
right -> down -> left -> up -> right -> down -> left -> up -> right -> right -> up -> left -> down -> down -> left
-> up -> right -> down -> left -> up -> right -> down -> left -> up -> up -> right -> down -> down -> left -> up
-> right -> down -> left -> up -> right -> down -> left -> up -> up -> right -> down -> down -> left -> up -> right
-> down -> left -> up -> right -> down -> right -> up -> left -> down -> left -> up -> right -> down -> left -> up
-> right -> down -> left -> up -> up -> right -> down -> down -> left -> up -> right -> down -> left -> up -> right
-> down -> left -> up -> up -> right -> down -> down -> left -> up -> right -> down -> left -> up -> right ->
down -> left -> up -> up -> right -> right -> down -> down -> left -> left -> up -> right -> down -> left -> up ->
right -> down -> left -> up -> right -> right -> up -> left -> down -> down -> left -> up -> up

Cost: 42111

Nodes Expanded: 43310

Search Depth: 42110

Running Time: 33.389141 seconds
```

**d.**



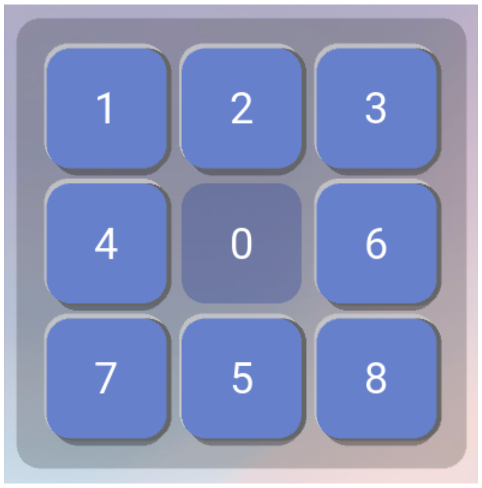


Puzzle Results

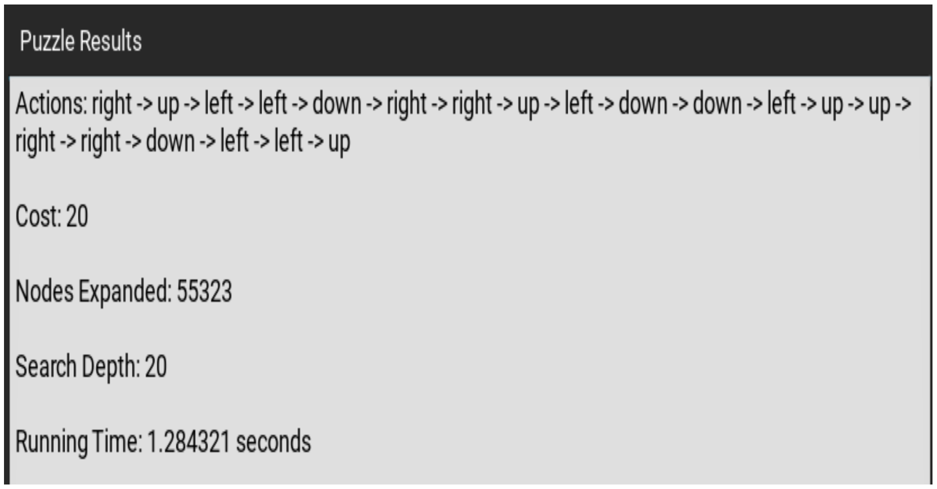
```
down -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> up -> right -> down -> down
-> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> up -> right -> right -> down -> down
-> left -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> right -> right -> down -> left
-> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> right -> right -> down -> left -> left
-> up -> right -> down -> left -> up -> right -> down -> left -> up -> right -> up -> right -> down -> down -> left
-> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> right -> right -> down -> left -> left
-> up -> right -> down -> left -> up -> right -> down -> left -> up -> right -> right -> down -> left -> left -> up ->
right -> down -> left -> up -> right -> down -> left -> up -> right -> up -> left -> down -> down -> right -> right
-> up -> left -> down -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> up -> right ->
down -> down -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> up -> right -> down
-> down -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> up -> right -> right -> down
-> down -> left -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> right -> right ->
down -> left -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> right -> right -> down
-> left -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> right -> up -> left -> down ->
down -> right -> right -> up -> left -> down -> left -> up -> right -> down -> left -> up -> right -> down -> left
-> up -> up -> right -> down -> down -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up
-> up -> right -> down -> down -> left -> up -> right -> down -> left -> up -> right -> down -> left -> up -> up
-> right -> right -> down -> down -> left -> left -> up -> right -> down -> left -> up -> up

Cost: 9634

Nodes Expanded: 9716

Search Depth: 9633

Running Time: 1.658994 seconds
```

**IDS:**

**a.**



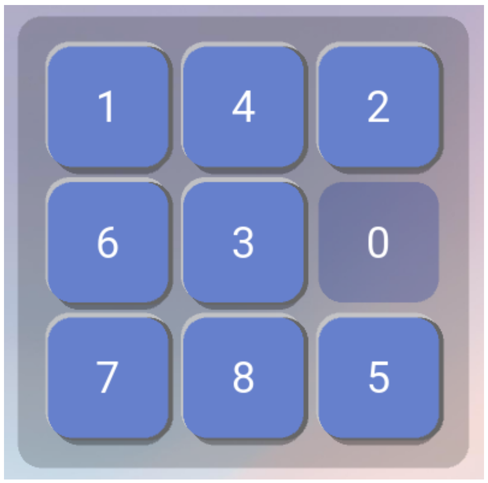
| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 0 | 6 |
| 7 | 5 | 8 |



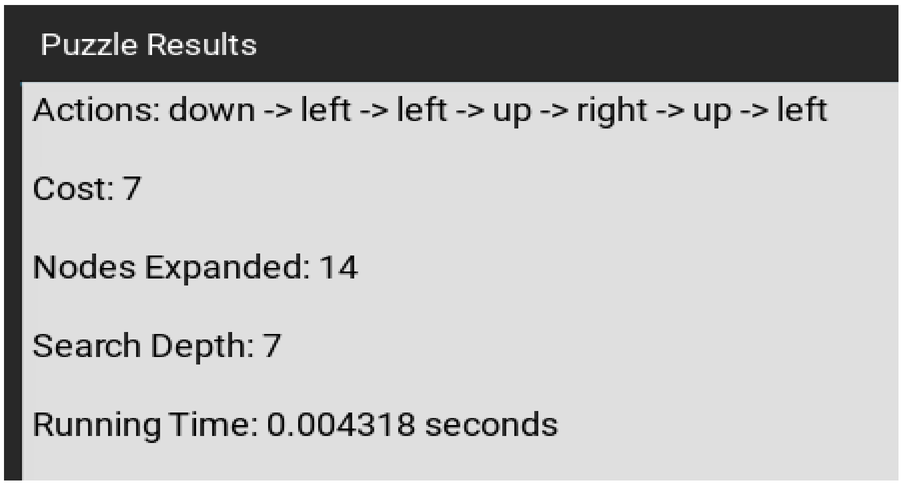
Puzzle Results

```
Actions: right -> up -> left -> left -> down -> right -> right -> up -> left -> down -> down -> left -> up -> up ->
right -> right -> down -> left -> left -> up

Cost: 20

Nodes Expanded: 55323

Search Depth: 20

Running Time: 1.284321 seconds
```

**b.**



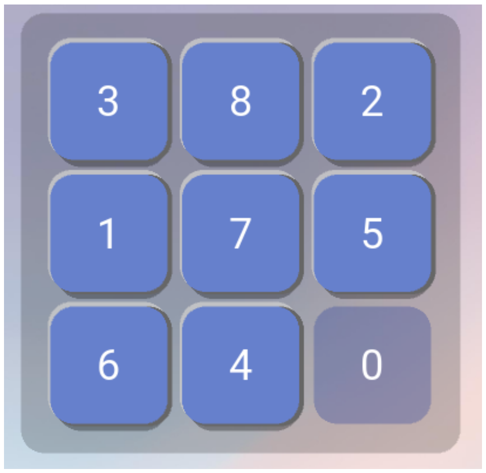
| | | |
|---|---|---|
| 1 | 4 | 2 |
| 6 | 3 | 0 |
| 7 | 8 | 5 |



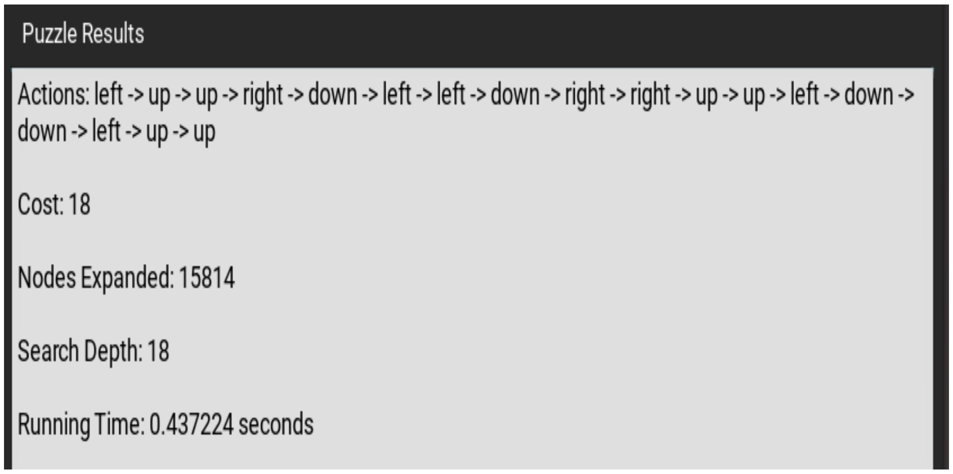
Puzzle Results

```
Actions: down -> left -> left -> up -> right -> up -> left

Cost: 7

Nodes Expanded: 14

Search Depth: 7

Running Time: 0.004318 seconds
```

**c.**



| | | |
|---|---|---|
| 3 | 8 | 2 |
| 1 | 7 | 5 |
| 6 | 4 | 0 |

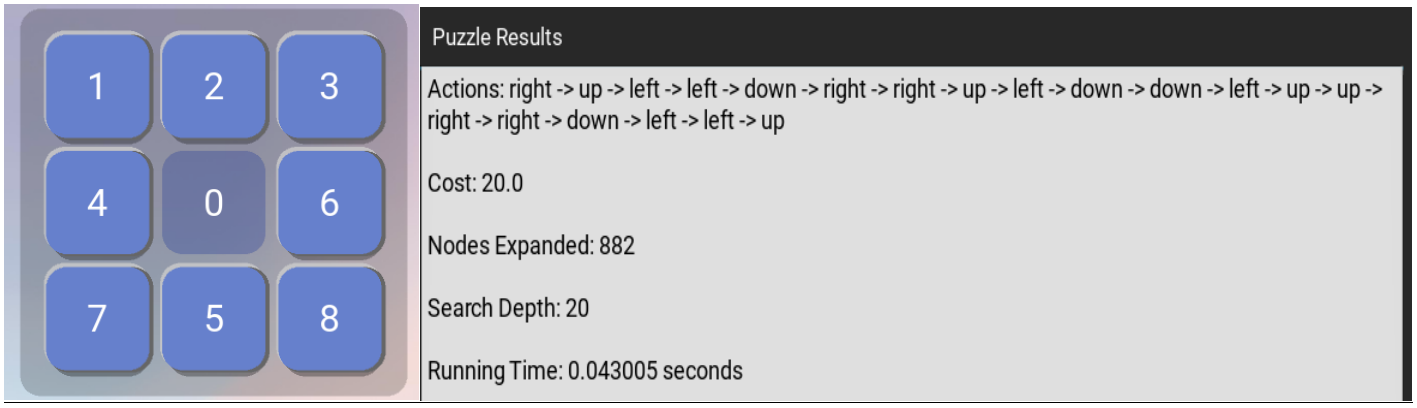


Puzzle Results

```
Actions: left -> up -> up -> right -> down -> left -> left -> down -> right -> right -> up -> up -> left -> down ->
down -> left -> up -> up

Cost: 18

Nodes Expanded: 15814

Search Depth: 18

Running Time: 0.437224 seconds
```

## A*(Manhattan):

**a.**



Actions: right -> up -> left -> left -> down -> right -> right -> up -> left -> down -> down -> left -> up -> up -> right -> right -> down -> left -> left -> up

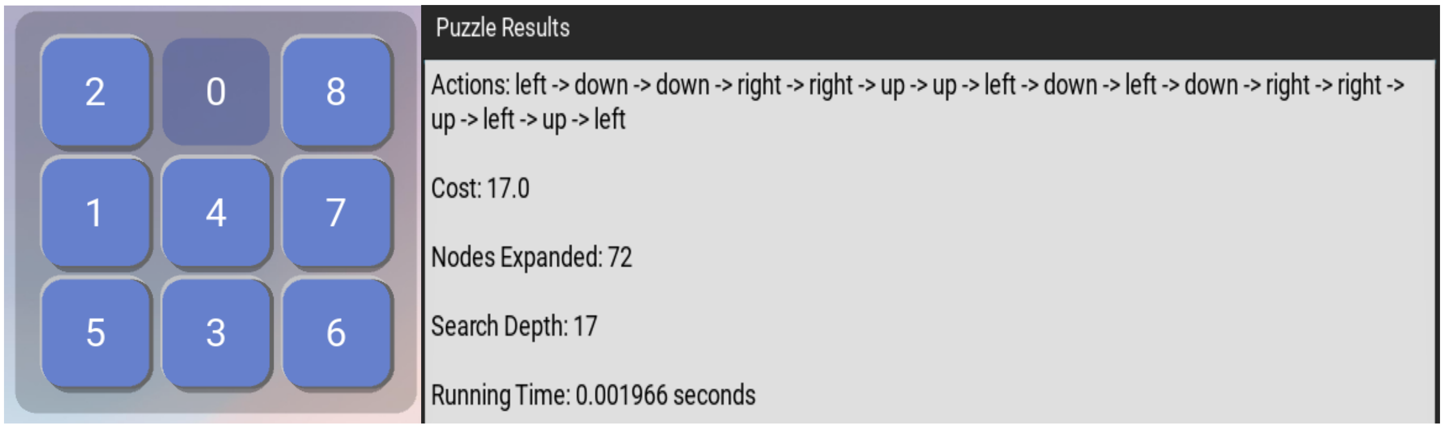
Cost: 20.0

Nodes Expanded: 882

Search Depth: 20

Running Time: 0.043005 seconds

**b.**



Actions: left -> down -> down -> right -> right -> up -> up -> left -> down -> left -> down -> right -> right -> up -> left -> up -> left

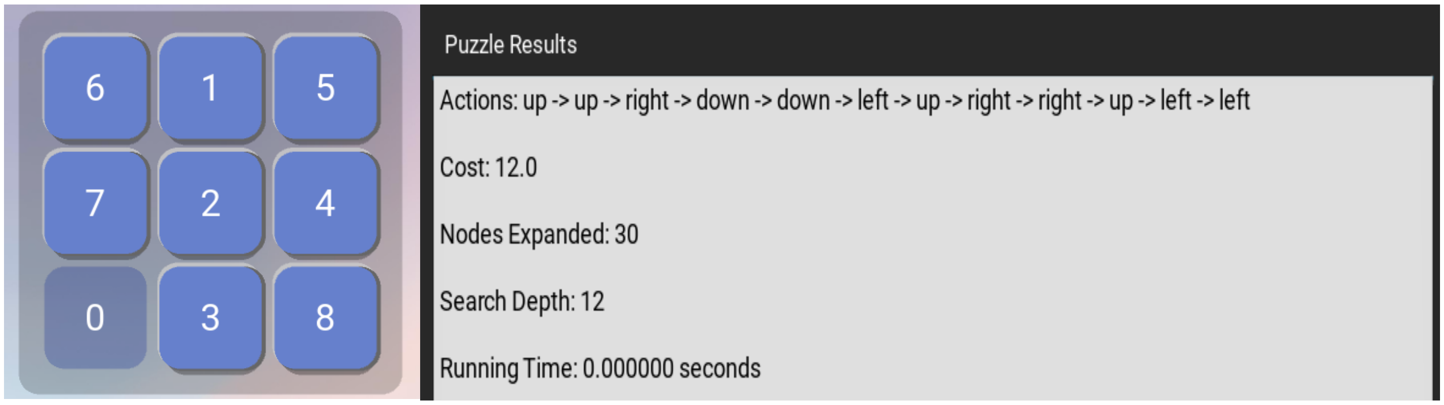
Cost: 17.0

Nodes Expanded: 72

Search Depth: 17

Running Time: 0.001966 seconds

**c.**



Actions: up -> up -> right -> down -> down -> left -> up -> right -> right -> up -> left -> left

Cost: 12.0

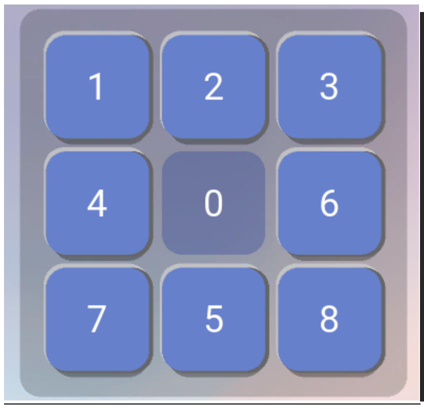
Nodes Expanded: 30

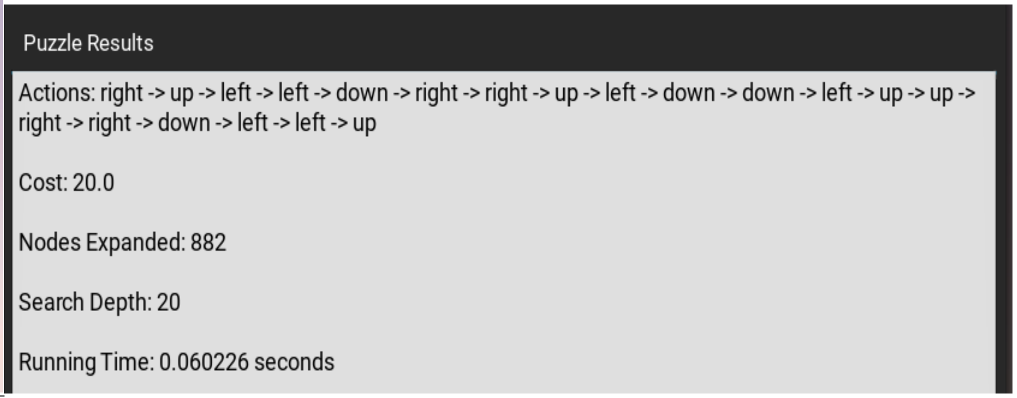
Search Depth: 12

Running Time: 0.000000 seconds

## A*(Euclidean):

**a.**




Puzzle Results

Actions: right -> up -> left -> left -> down -> right -> right -> up -> left -> down -> down -> left -> up -> up -> right -> right -> down -> left -> left -> up

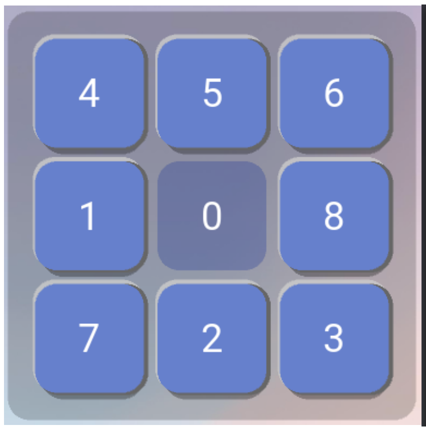
Cost: 20.0

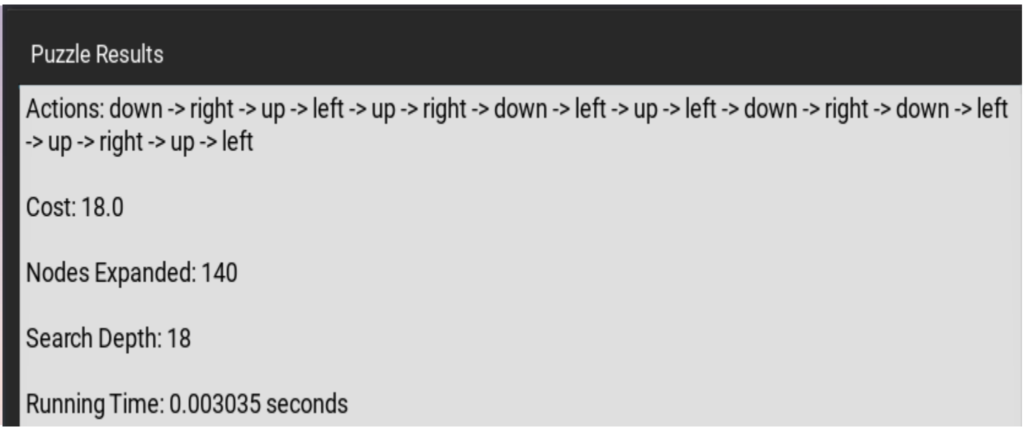
Nodes Expanded: 882

Search Depth: 20

Running Time: 0.060226 seconds

**b.**




Puzzle Results

Actions: down -> right -> up -> left -> up -> right -> down -> left -> up -> left -> down -> right -> down -> left -> up -> right -> up -> left

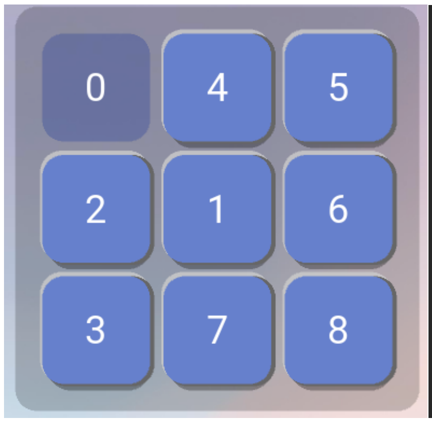
Cost: 18.0

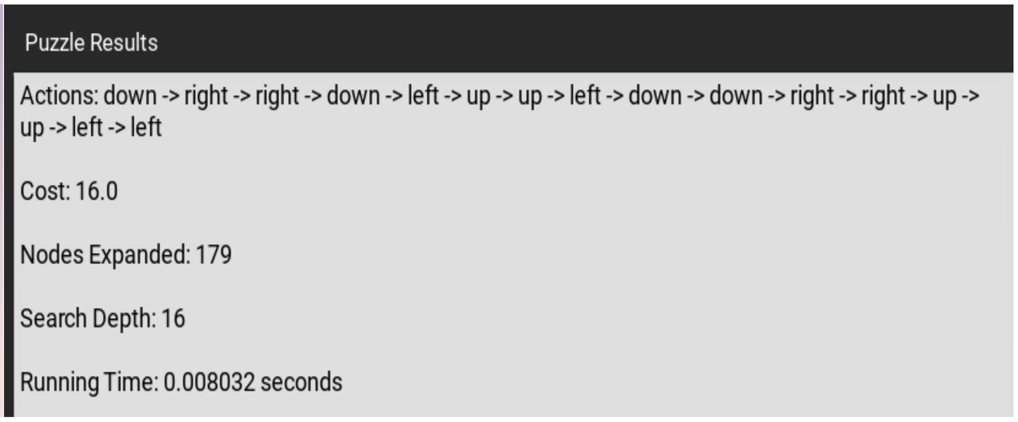
Nodes Expanded: 140

Search Depth: 18

Running Time: 0.003035 seconds

**c.**




Puzzle Results

Actions: down -> right -> right -> down -> left -> up -> up -> left -> down -> down -> right -> right -> up -> up -> left -> left

Cost: 16.0

Nodes Expanded: 179

Search Depth: 16

Running Time: 0.008032 seconds