Assignment 1

Abdelrahman Ibrahim Aly Hagrass 34
Abdelrahman Mahmoud Kamal Mahmoud Nour 37

Problem Statement:

An instance of the 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. In this assignment, the blank space is going to be represented with the number0. Given an initial state of the board, the search problem is to find a sequence of moves that transitions this state to the goal state; that is, the configuration with all tiles arranged in ascending order 0,1,2,3,4,5,6,7,8. The search space is the set of all possible states reachable from the initial state. The blank space may be swapped with a component in one of the four directions ‘Up’, ‘Down’, ‘Left’, ‘Right’, one move at a time. The cost of moving from one configuration of the board to another is the same and equal to one. Thus, the total cost of a path is equal to the number of moves made from the initial state to the goal state.

Data Structures:

We used **C++ STL** library data structure such as:

- Priority Queues.
- Queues.
- Pairs.
- Vectors.
- unordered_map

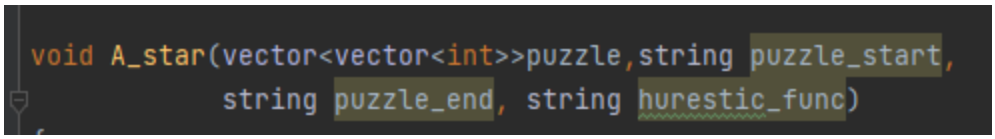
Algorithms:

We Implemented The three 3 required algorithms to solve the 8-puzzle which are:

- BFS.
- DFS.
- A* using Euclidean distance as a heuristic.
- A* using Manhattan distance as a heuristic.

Functions Implemented:

- void print_state(int n,int m,string puzzle_str)
- pair<int, int> find_zero(vector<vector<int>>puzzle)
-
```
void add_states_to_queue(int n,int m,string state_str,
                         int level,
                         pair<int,int>pos_pair,
                         queue<pair<string,int>>&states,
                         queue<pair<int,int>>&zero_pos,
                         unordered_map<string,pair<string,string>>&parent,
                         unordered_map<string,bool>vis,
                         int& expanded)
```
- void bfs(vector<vector<int>>puzzle,string start_state_str,string finish_state_str)
-
```
void dfs_solver(int n,int m,string puzzle_str,
                pair<int, int>zero_pos,
                string final_state,
                vector<string>&moves,
                vector<string>&path,
                unordered_map<string,bool>&vis,
                bool &found,
                int &depth,int level,int &nodes_expanded)
```
- void dfs(vector<vector<int>>puzzle,string start_state, string final_state)
- float manhattan_dist(int n,int m,string puzzle_str)
- float ecludian_dist(int n, int m, string puzzle_str)

- 
```
void A_star(vector<vector<int>>puzzle,string puzzle_start,
            string puzzle_end, string hurestic_func)
```
- int getInvCount(string arr)
- bool isSolvable(string puzzle)

Assumptions and Details:

- Input is taken at runtime.
- Specification of A* algorithm heuristic is hardcoded.
- Path to goal and all the required measures are printed.
- Solvability is detected, Where if the number of inversions is equal to odd the puzzle cannot be solved else it can be solved.

Sample Runs:

Test Case 1: 1 2 5 3 4 0 6 7 8

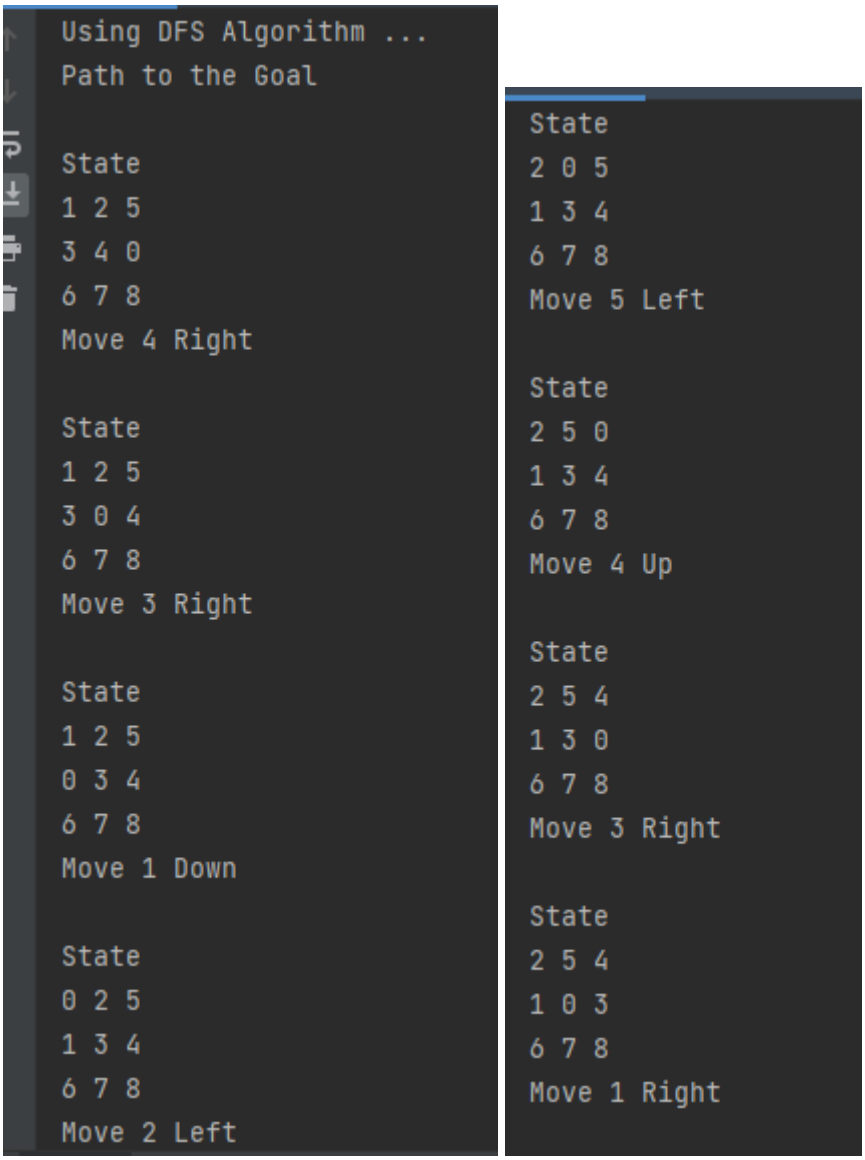
1	2	5
3	4	
6	7	8

BFS:

```
Using BFS Algorithm ...
Path to Goal

State
1 2 5
3 4 0
6 7 8
Move 5 Down

State
1 2 0
3 4 5
6 7 8
Move 2 Right

State
1 0 2
3 4 5
6 7 8
Move 1 Right
```

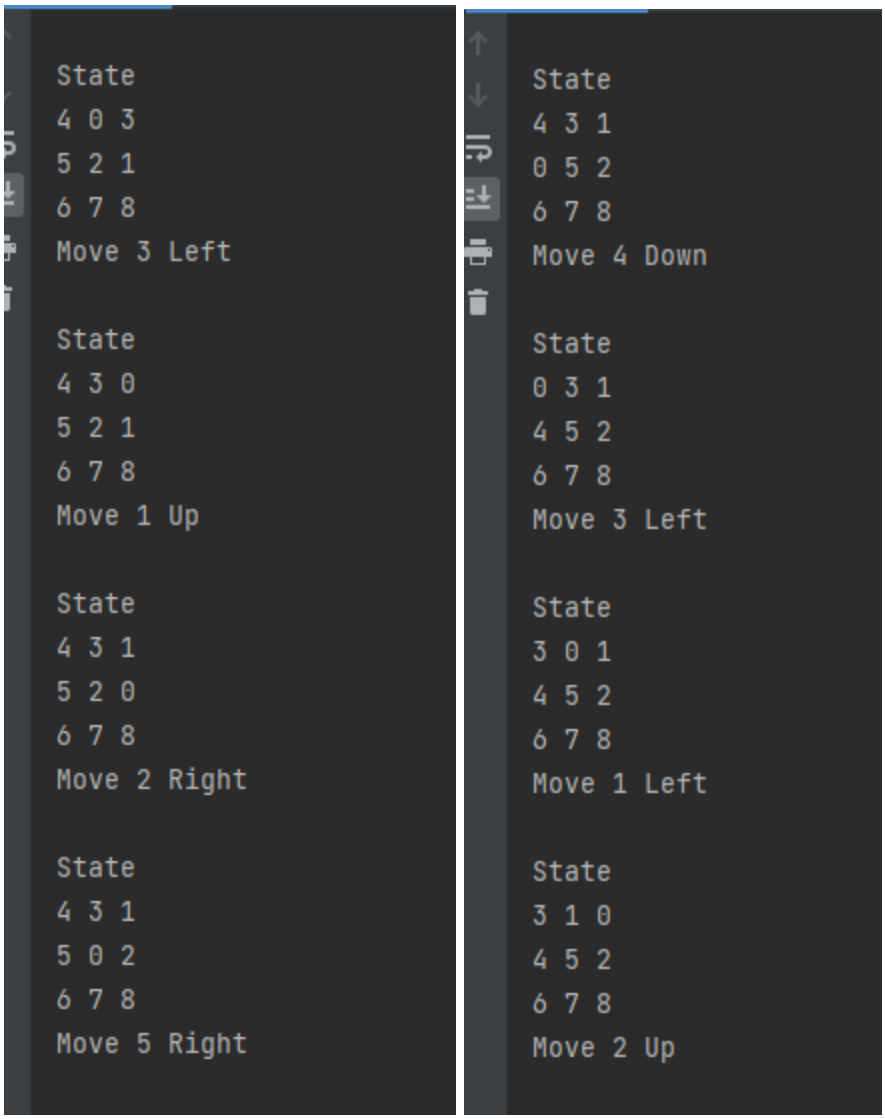
```
State
0 1 2
3 4 5
6 7 8

Depth: 3
Cost: 3
Node expanded: 25

9ms
------------------------
```

DFS:


Using DFS Algorithm ...
Path to the Goal
State
1 2 5
3 4 0
6 7 8
Move 4 Right
State
1 2 5
3 0 4
6 7 8
Move 3 Right
State
1 2 5
0 3 4
6 7 8
Move 1 Down
State
0 2 5
1 3 4
6 7 8
Move 2 Left
State
2 0 5
1 3 4
6 7 8
Move 5 Left
State
2 5 0
1 3 4
6 7 8
Move 4 Up
State
2 5 4
1 3 0
6 7 8
Move 3 Right
State
2 5 4
1 0 3
6 7 8
Move 1 Right

```
State
2 5 4
0 1 3
6 7 8
Move 2 Down

State
0 5 4
2 1 3
6 7 8
Move 5 Left

State
5 0 4
2 1 3
6 7 8
Move 4 Left

State
5 4 0
2 1 3
6 7 8
Move 3 Up
```

```
State
5 4 3
2 1 0
6 7 8
Move 1 Right

State
5 4 3
2 0 1
6 7 8
Move 2 Right

State
5 4 3
0 2 1
6 7 8
Move 5 Down

State
0 4 3
5 2 1
6 7 8
Move 4 Left
```


State
4 0 3
5 2 1
6 7 8
Move 3 Left
State
4 3 0
5 2 1
6 7 8
Move 1 Up
State
4 3 1
5 2 0
6 7 8
Move 2 Right
State
4 3 1
5 0 2
6 7 8
Move 5 Right
State
4 3 1
0 5 2
6 7 8
Move 4 Down
State
0 3 1
4 5 2
6 7 8
Move 3 Left
State
3 0 1
4 5 2
6 7 8
Move 1 Left
State
3 1 0
4 5 2
6 7 8
Move 2 Up

10

```
State
3 1 2
4 5 0
6 7 8
Move 5 Right

State
3 1 2
4 0 5
6 7 8
Move 4 Right

State
3 1 2
0 4 5
6 7 8
Move 3 Down

State
0 1 2
3 4 5
6 7 8
```

```
Depth: 27
Cost: 27
Nodes expanded: 27

69ms
--------------------------------
```

A* using Euclidean heuristic.

```
Using A* Algorithm using Euclidean Distance ...
Path to the goal

State
1 2 5
3 4 0
6 7 8
Move 5 Down

State
1 2 0
3 4 5
6 7 8
Move 2 Right

State
1 0 2
3 4 5
6 7 8
Move 1 Right

State
0 1 2
3 4 5
6 7 8
```

```
6 7 8

Node expanded: 6
Depth : 3
Cost : 3

9ms
```

A* using Manhattan heuristic.

```
Using A* Algorithm using Manhattan Distance ...
Path to the goal

State
1 2 5
3 4 0
6 7 8
Move 5 Down

State
1 2 0
3 4 5
6 7 8
Move 2 Right

State
1 0 2
3 4 5
6 7 8
Move 1 Right

State
0 1 2
3 4 5
6 7 8
```

```
6 7 8

Node expanded: 6
Depth : 3
Cost : 3

9ms
```

	BFS	DFS	A*(Euclidean)	A*(Manhattan)
Time	9 ms	69ms	9ms	9ms
Nodes Expanded	25	27	6	6
Depth	3	27	3	3
Cost	3	27	3	3

Test Case 2:

1 0 2

3 4 5

6 7 8

(Simple Example)

	BFS	DFS	A*(Euclidean)	A*(Manhattan)
Time	0 ms	9ms	0ms	0ms
Nodes Expanded	7	1	3	3
Depth	1	1	1	1
Cost	1	1	1	1

```
Using BFS Algorithm ...
Path to Goal

State
1 0 2
3 4 5
6 7 8
Move 1 Right

State
0 1 2
3 4 5
6 7 8

Depth: 1
Cost: 1
Node expanded: 7

0ms
--------------------------------
```

```
Using DFS Algorithm ...
Path to the Goal

State
1 0 2
3 4 5
6 7 8
Move 1 Right

State
0 1 2
3 4 5
6 7 8

Depth: 1
Cost: 1
Nodes expanded: 1

9ms
--------------------------------
```

```
---------------------------------

Using A* Algorithm using Euclidean Distance ...
Path to the goal

State
1 0 2
3 4 5
6 7 8
Move 1 Right

State
0 1 2
3 4 5
6 7 8

Node expanded: 3
Depth : 1
Cost : 1

0ms
-----------------------------------
```

```
---------------------------------

Using A* Algorithm using Manhattan Distance ...
Path to the goal

State
1 0 2
3 4 5
6 7 8
Move 1 Right

State
0 1 2
3 4 5
6 7 8

Node expanded: 3
Depth : 1
Cost : 1

0ms
```

Test Case 3:

```
1  2  3
4  5  6
8  7
```

Is unsolvable because it has an odd number of inversions.