Faculty of Engineering, Alexandria University

Computer and Systems Engineering Department

Aritficial Intelligence: Fall 2018

Assignment 1

Using Informed and Uninformed Search Algorithms to Solve 8-Puzzle

	Name	SID
(1)	Aya Aly Abouzeid	2
(2)	Amira Nabil Shaaban	19
(3)	Yosef Zook	90

Data structures :

- ***State :*** an object containing
  - 2D array resembling the game grid.
  - Point saving the zero index.
  - Arraylist of neighboring states.
  - A pointer to my parent state.
  - A string showing the direction that lead to this state.

```
private int game[][] = new int[3][3];
private Point zeroPosition;
private ArrayList<State> neighbours = new ArrayList<>();
private State parent;
private String path;
```

- ***AstarState*** : an object wrapper for any state in A* algorithm
  - State object
  - H(n)
  - G(n)
  - Total Cost = H(n) + G(n)

```
private State state;
private int g, h, cost;
```

- ***ArrayList*** : for explored states.
- ***PriorityQueue*** : for frontier List in A* algorithm (sort according to the cost).
- ***Queue*** : for frontier List in BFS algorithm.
- ***Stack*** : for frontier List in A* algorithm (sort according to the cost).

Algorithms:

BFS:

- Search level by level to reach the goal
- takes huge space if the goal is at the least levels $O(b^d)$
- take exponential time $O(b^d)$
- Solution is optimal

```
Queue<State> frontier = new LinkedList<State>();
frontier.push(startState);
while (!frontier.isEmpty()) {
        State s = frontier.remove();
        explored.add(s);
        if (s.testGoal()) {
                stopTime = System.currentTimeMillis();
                elapsedTime = stopTime - startTime;
                finalState = s;
                return true;
        }
        for (State neighbour : s.getNeighbours()) {
                if (!isExplored(neighbour, explored) && !inFrontier(neighbour, frontier)) {
                        frontier.add(neighbour);
                }
        }
}
stopTime = System.currentTimeMillis();
elapsedTime = stopTime - startTime;
return false;
```

DFS:

- Search in each branch till the end of it before going to the next one
- takes huge time if the goal is at the left branches $O(b^m)$
- take less memory than BFS O(bm)
- Solution is not necessarily optimal

```
Stack<State> frontier = new Stack<>();

frontier.push(startState);

while (!frontier.isEmpty()) {
        State s = frontier.pop();
        explored.add(s);

        if (s.testGoal()) {
                stopTime = System.currentTimeMillis();
                elapsedTime = stopTime - startTime;
                finalState = s;
```

```
                        return true;
                }

                ArrayList<State> neighbours = s.getNeighbours();
                for (State neighbour : neighbours)
                        if (!inFrontier(neighbour, frontier) && !inExplored(neighbour, explored))
                                frontier.push(neighbour);

        }
        stopTime = System.currentTimeMillis();
        elapsedTime = stopTime - startTime;
        return false;
```

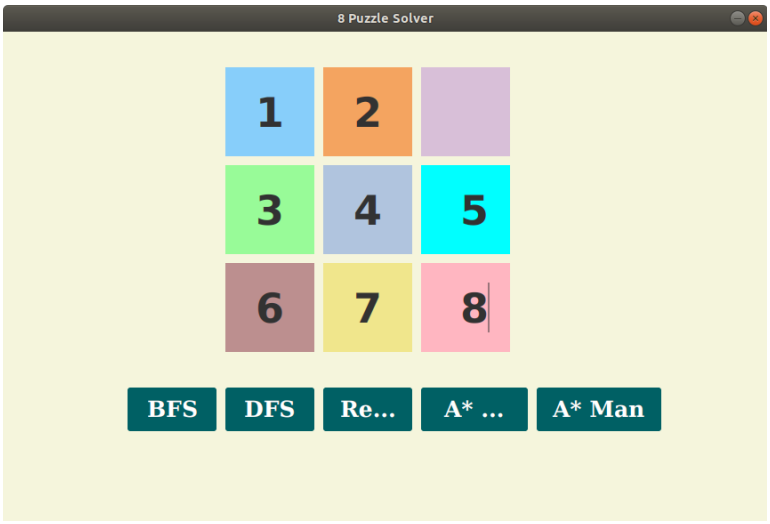
A* Search:

- It has an exponential running time.
- For each tree level, G(n) = G(n-1)+1
- For each state, H(n) is computed according to the chosen heuristic.
- We add the visited states to the explored list and their valid neighbours to the frontier list, while the frontier list has states we keep of visiting the least cost state until we get the goal state.

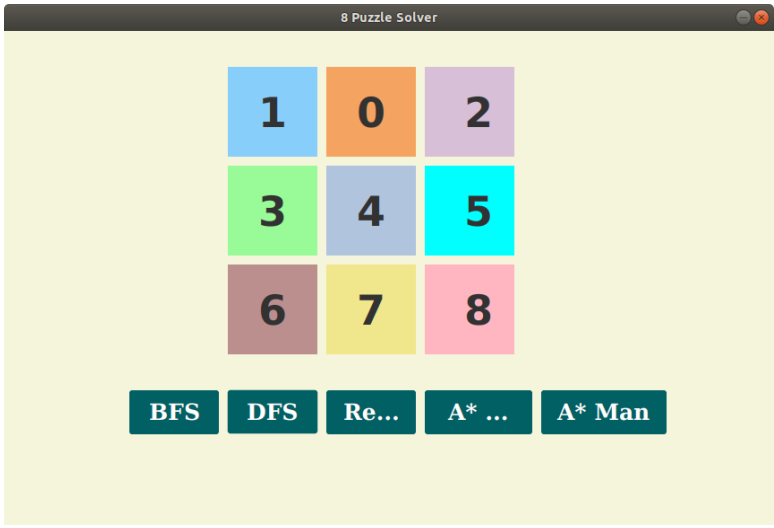
Comparing H(n) computing strategies:
Manhattan heuristic is an admissible heuristic where it never over estimates the real path cost from the current state to the goal state where Euclidean heuristic does.

(a) path to goal: Each state has a reference where it came from.
(b) cost of path: G(n) increases by one as the tree goes deeper.
(c) nodes expanded: Size of explored list.
(d) search depth: levels of the tree.
(e) running time: Using System.currentTimeMillis() before and after the algorithm's execution and getting the difference between them.

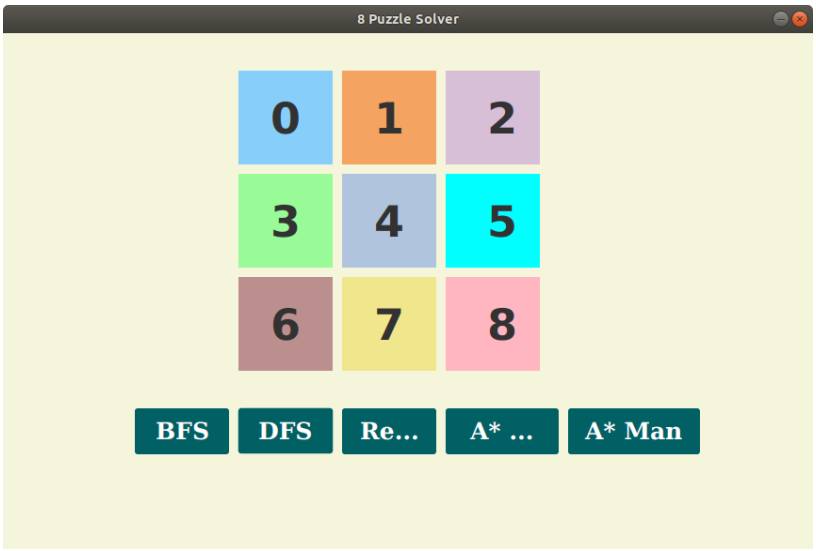
Sample Runs :



Step 0 - start state

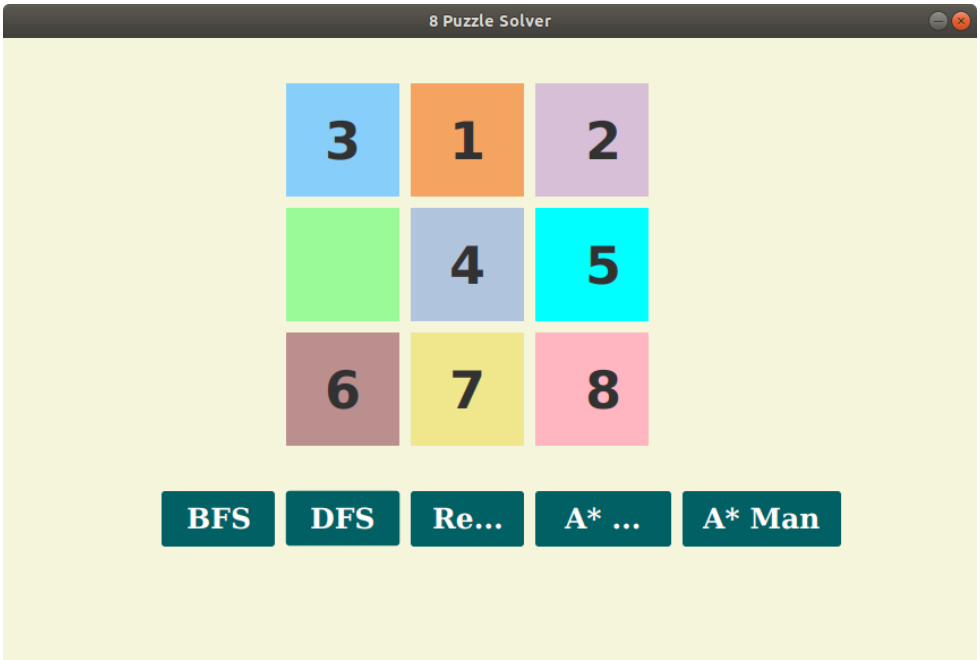


step 1

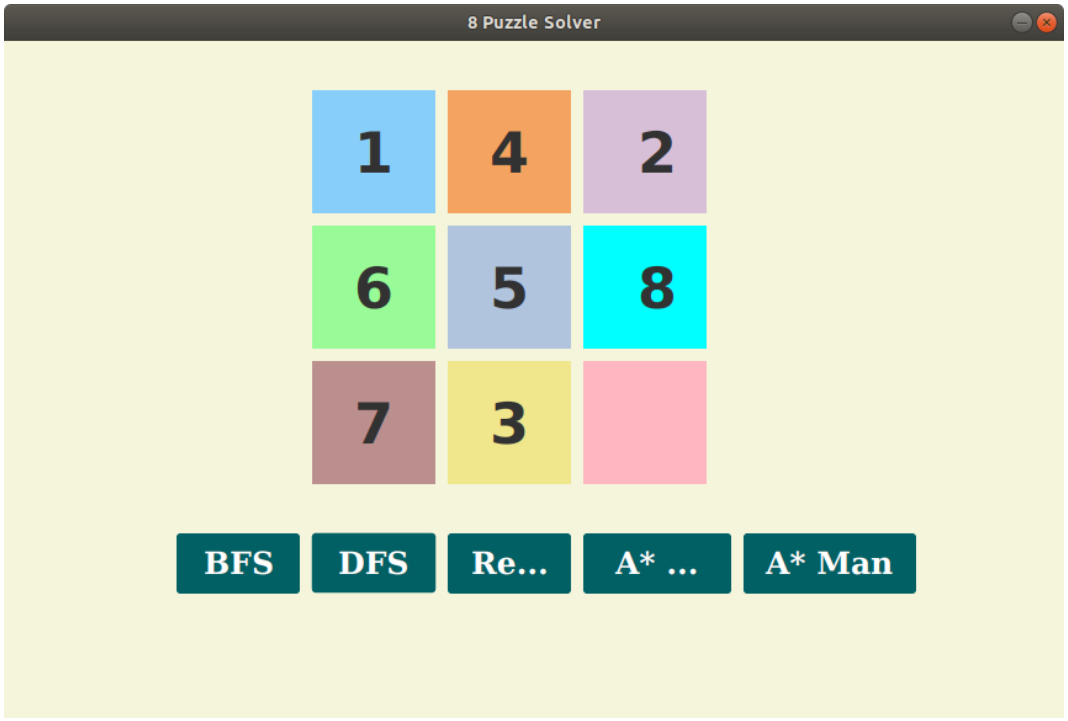


step 2 - goal state

	DFS	BFS	A* man	A* euc
Path Cost	2	2	2	2
Nodes Expanded	3	7	3	3
Path to goal	Left Left	Left Left	Left Left	Left Left
Running Time (s)	0	0	0	0
depth	2	2	2	2



	DFS	BFS	A* man	A* euc
Path Cost	1	1	1	1
Nodes Expanded	181439	3	2	2
Path to goal	Up	Up	Up	Up
Running Time (s)	1321664	0	0	0
depth	1	1	1	1



	DFS	BFS	A* man	A* euc
Path Cost	541	8	8	8
Nodes Expanded	151392	195	12	11
Path to goal	Huge	Up Left Down Left Up Right Up Left	Up Left Down Left Up Right Up Left	Up Left Down Left Up Right Up Left
Running Time (s)	24 min	1	1	1
depth	542	8	8	8