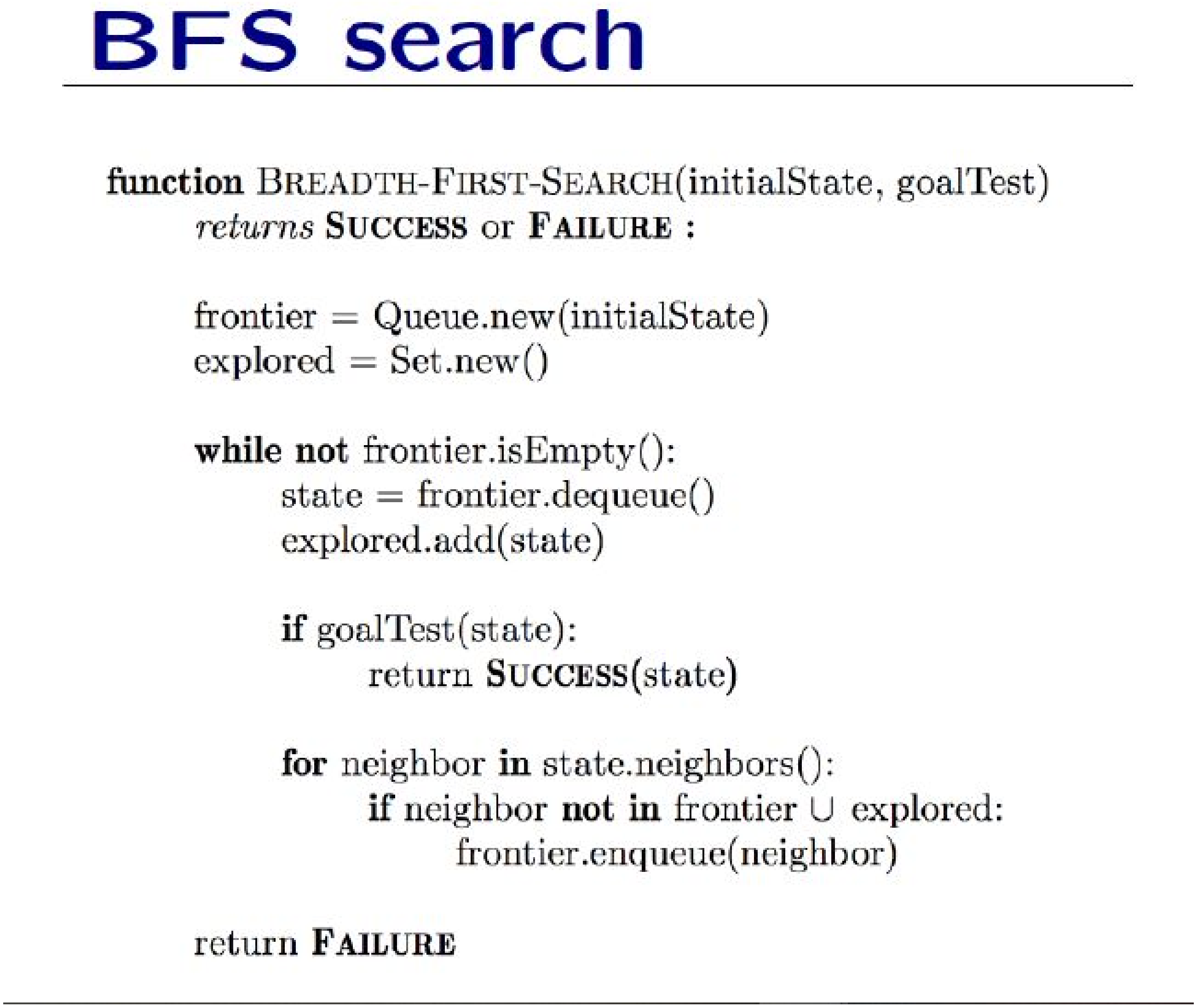
AI Assignment 1

Using Informed and Uninformed Search Algorithms to Solve 8-Puzzle

Team Members

- Ashraqat Sheta (13)
- Fatma Ibrahim (42)
- Marina Zakaria (47)

1. BFS Algorithm



BFS search

```
function BREADTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Queue.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.dequeue()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier $\cup$ explored:
                frontier.enqueue(neighbor)

    return FAILURE
```

2. DFS Algorithm

DFS search

```
function DEPTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Stack.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.pop()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier $\cup$ explored:
                frontier.push(neighbor)

    return FAILURE
```

Used Data Structure and Classes :

Class State : contains a) the state cells in 2 dimensional array b) cost value 3) heuristic value

- LinkedList <State> : Used to store path ,expanded nodes and states expected to be visited
  Output : Absolute path
  Explored : expanded nodes
  Frontier : contains the states that are expected to be visited
- LinkedList <LinkedList<Integer>> :
  Id : used to keep track of parents of each node

Utiles :

- int get_index (int x, int array[][],int flag)
  This function takes element and a state and return the index of that element in the given state
  Return x index if flag = 0 and y index otherwise

- int[][] swap(int cell1 , int cell2 ,int array[][])
  This function takes two adjacent elements and a state swap the two elements and returns
  A neighbouring state

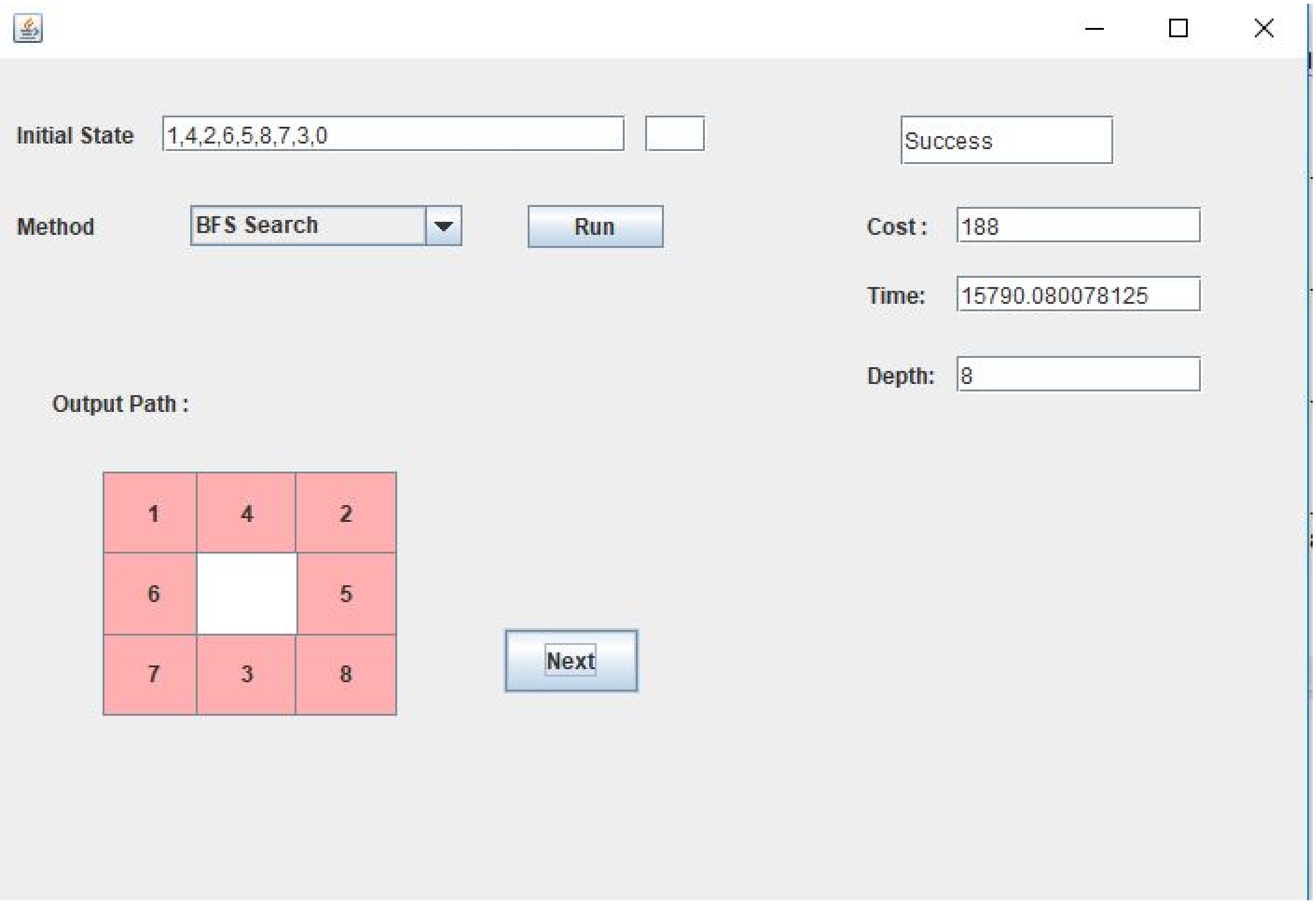
- boolean compare (int a1[][], int a2[][])
  This function compares a two state return true if they are the same and false if not

- LinkedList<State> get_neibours(State stateObj)
  This function returns the surrounding neighbours from all directions

- boolean contain(LinkedList<State>list,State state)
  This function checks if a selected state is contained in a list or not

Sample Run:

1. BFS



Initial State 1,4,2,6,5,8,7,3,0

Method BFS Search

Success

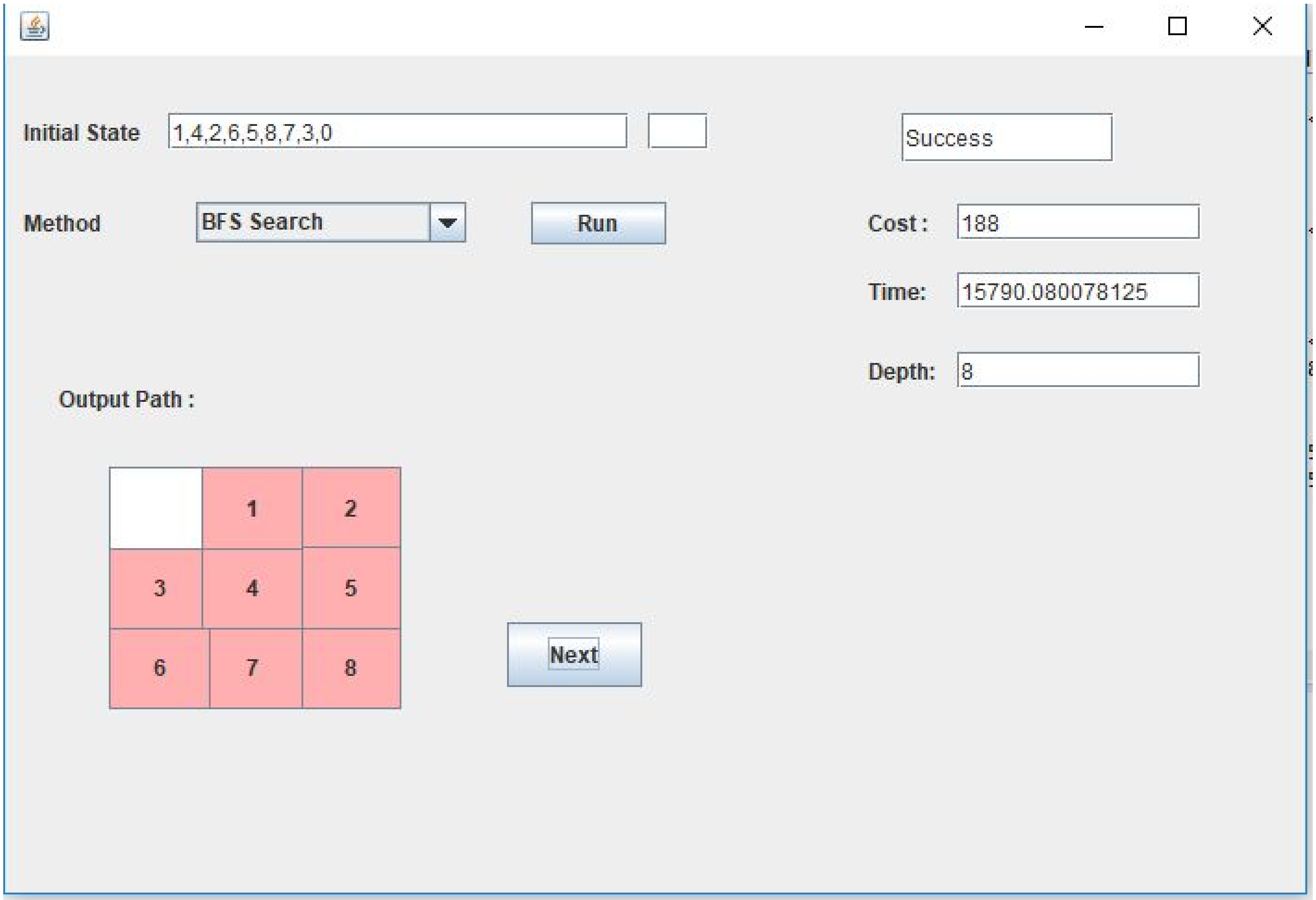
Cost : 188

Time: 15790.080078125

Depth: 8

Output Path :

1	4	2
6		5
7	3	8



Initial State 1,4,2,6,5,8,7,3,0

Method BFS Search

Success

Cost : 188

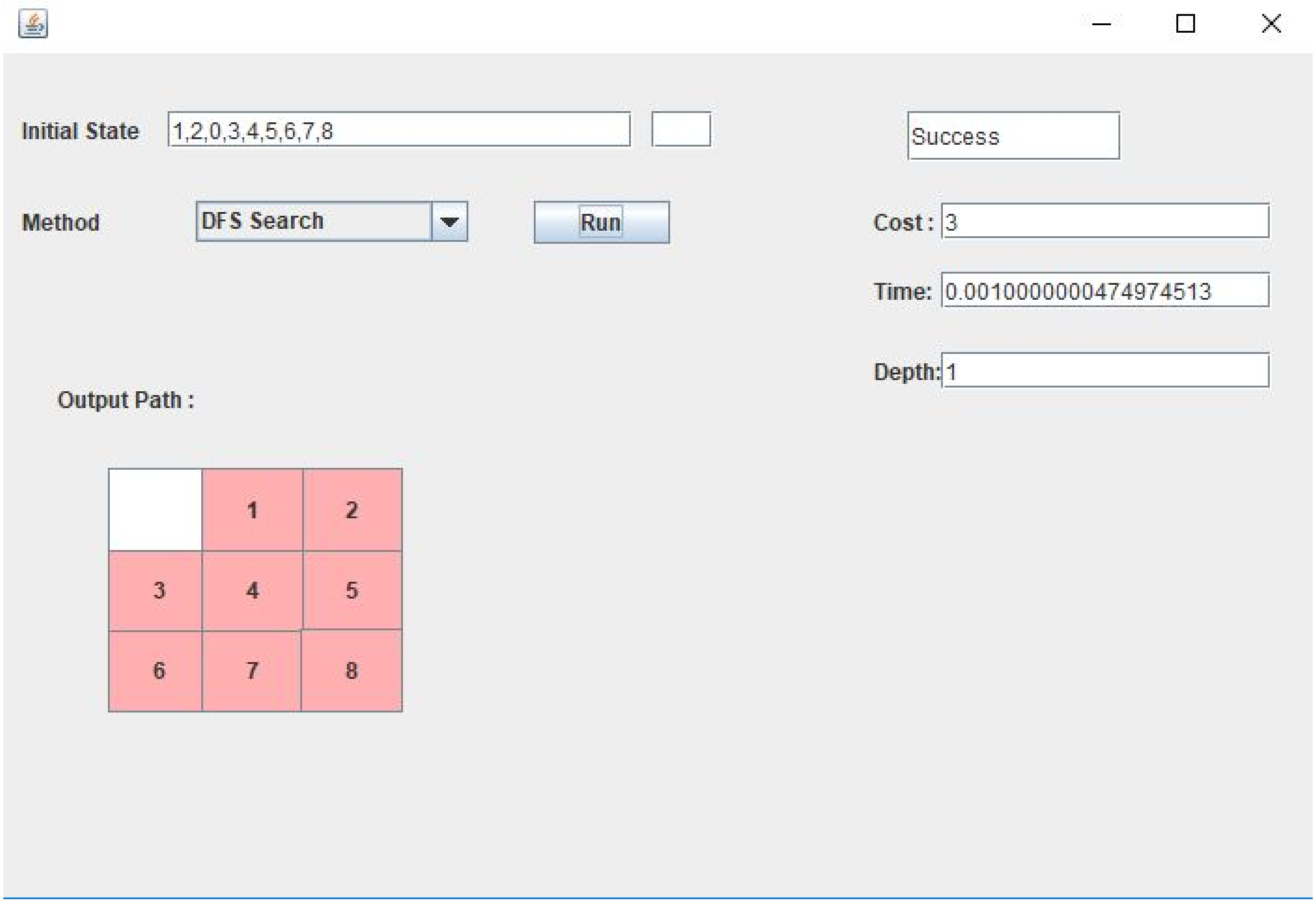
Time: 15790.080078125

Depth: 8

Output Path :

	1	2
3	4	5
6	7	8

2. DFS



Initial State 1,2,0,3,4,5,6,7,8

Success

Method DFS Search

Cost : 3

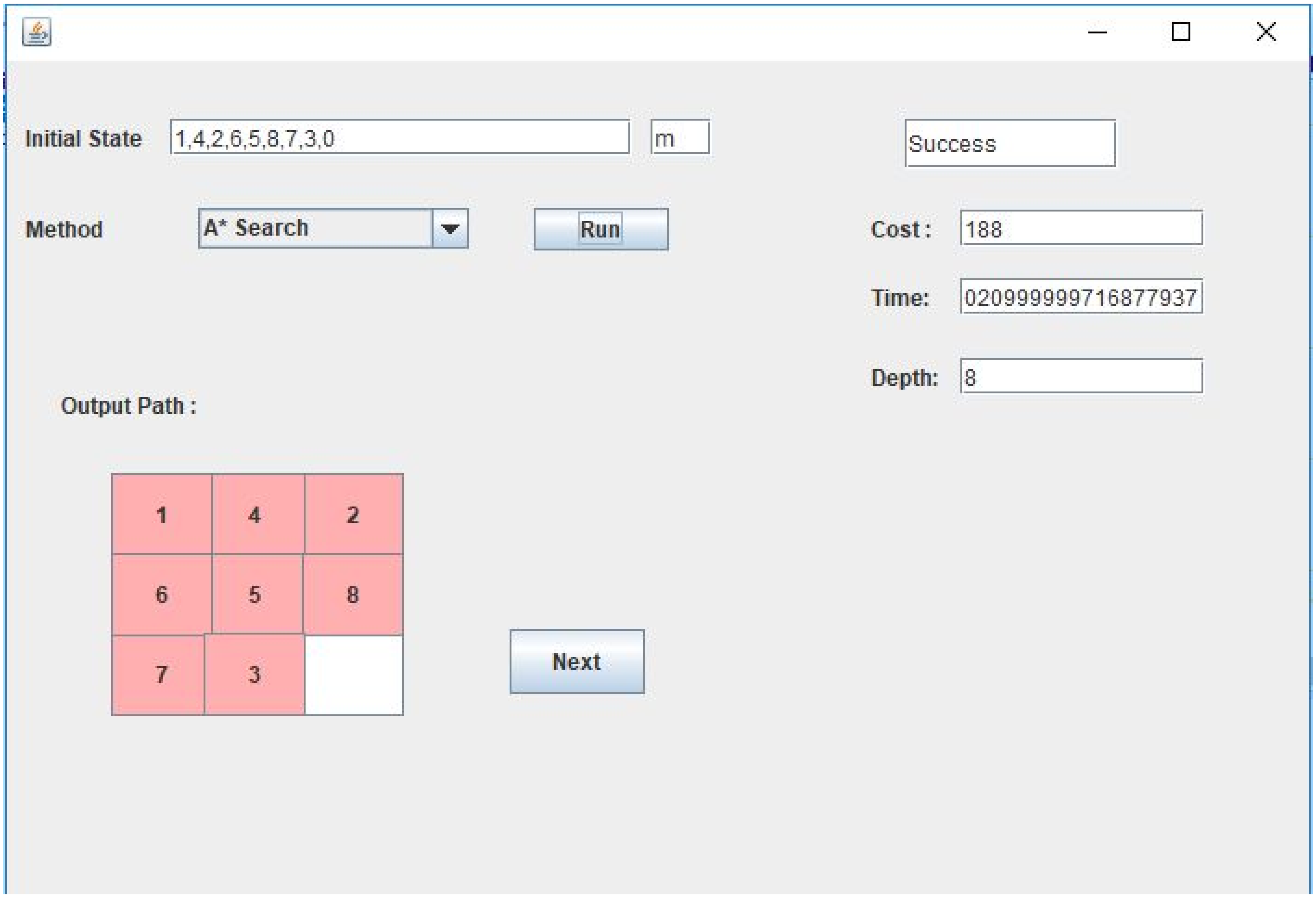
Time: 0.0010000000474974513

Depth: 1

Output Path :

	1	2
3	4	5
6	7	8

3. A*



Initial State 1,4,2,6,5,8,7,3,0 m

Success

Method A* Search

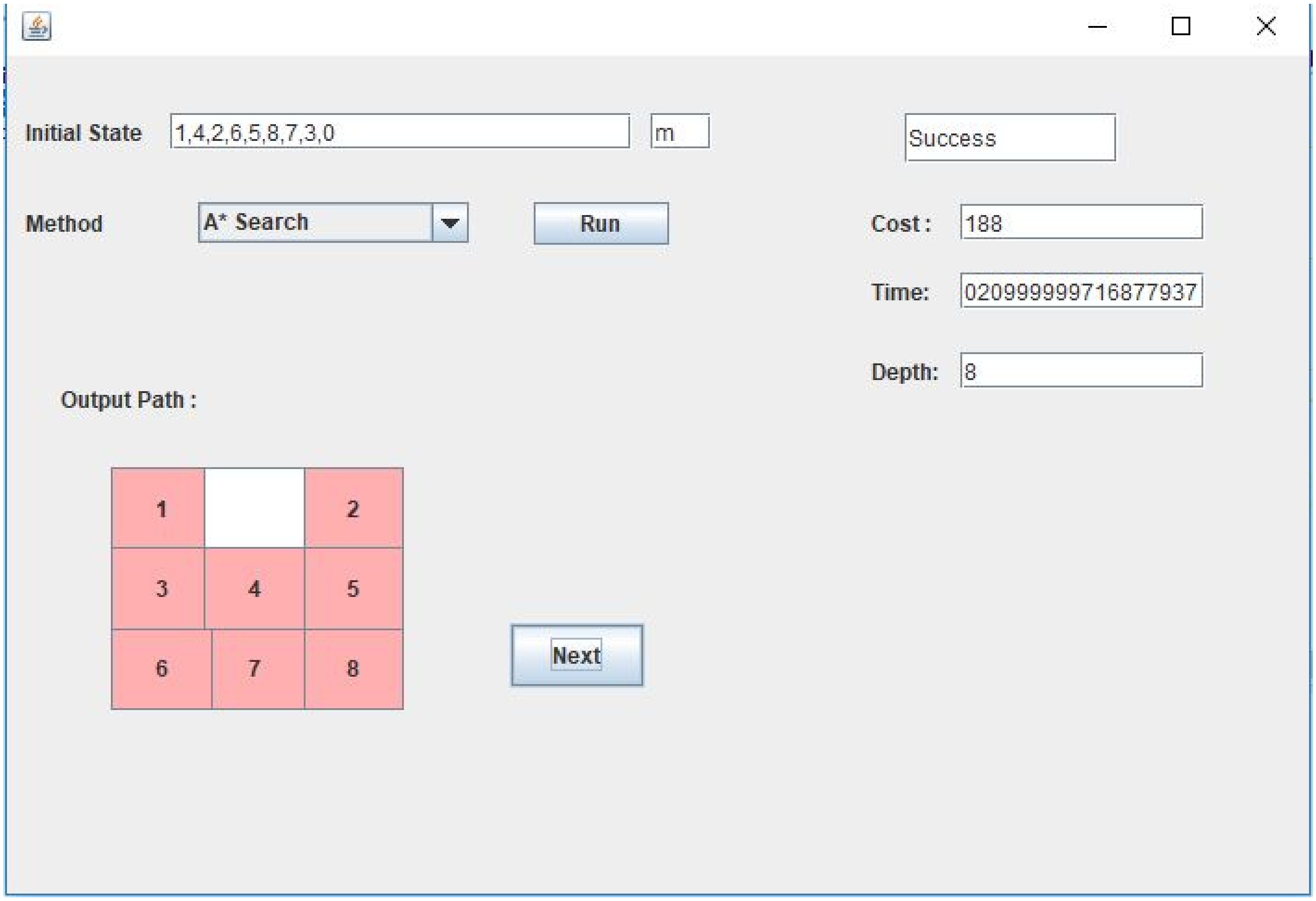
Cost : 188

Time: 020999999716877937

Depth: 8

Output Path :

1	4	2
6	5	8
7	3	



Initial State 1,4,2,6,5,8,7,3,0 m

Success

Method A* Search

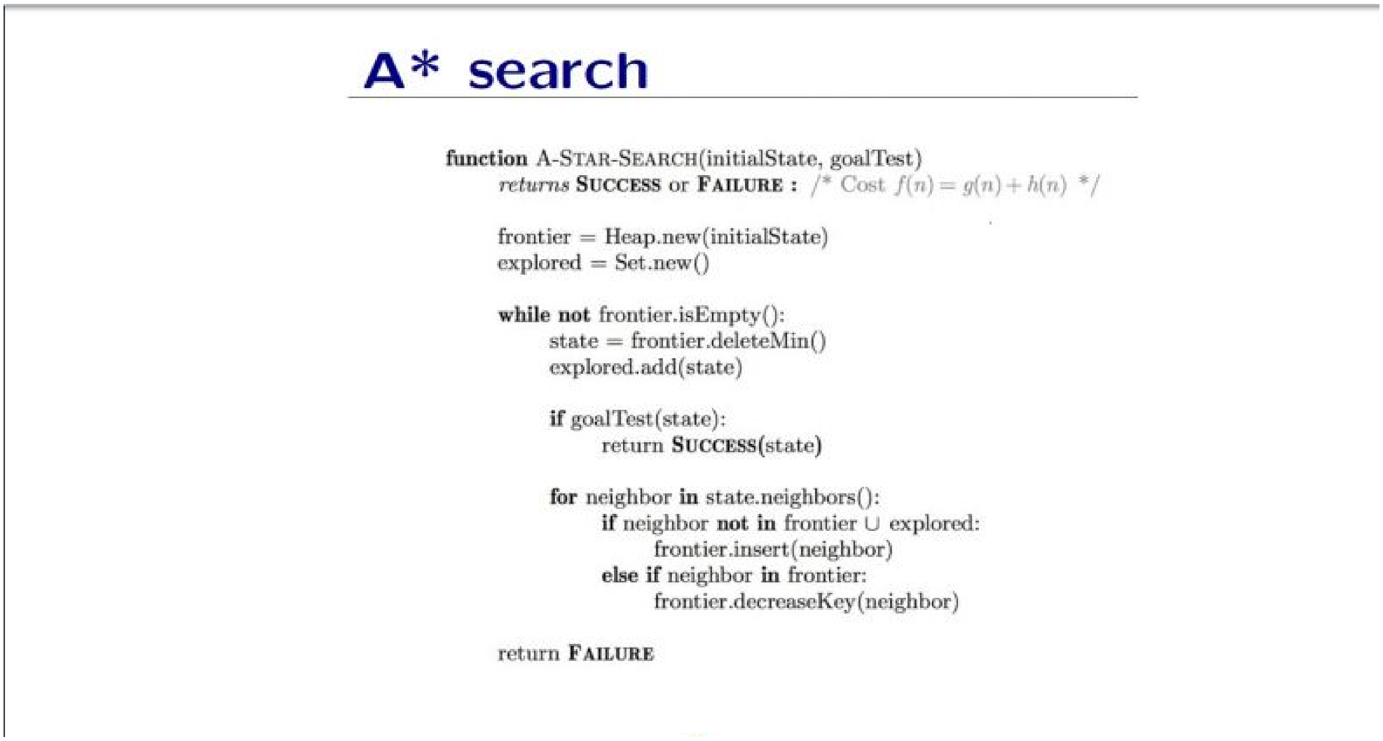
Cost : 188

Time: 020999999716877937

Depth: 8

Output Path :

1		2
3	4	5
6	7	8



A* search

```
function A-STAR-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE : /* Cost $f(n) = g(n) + h(n)$ */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

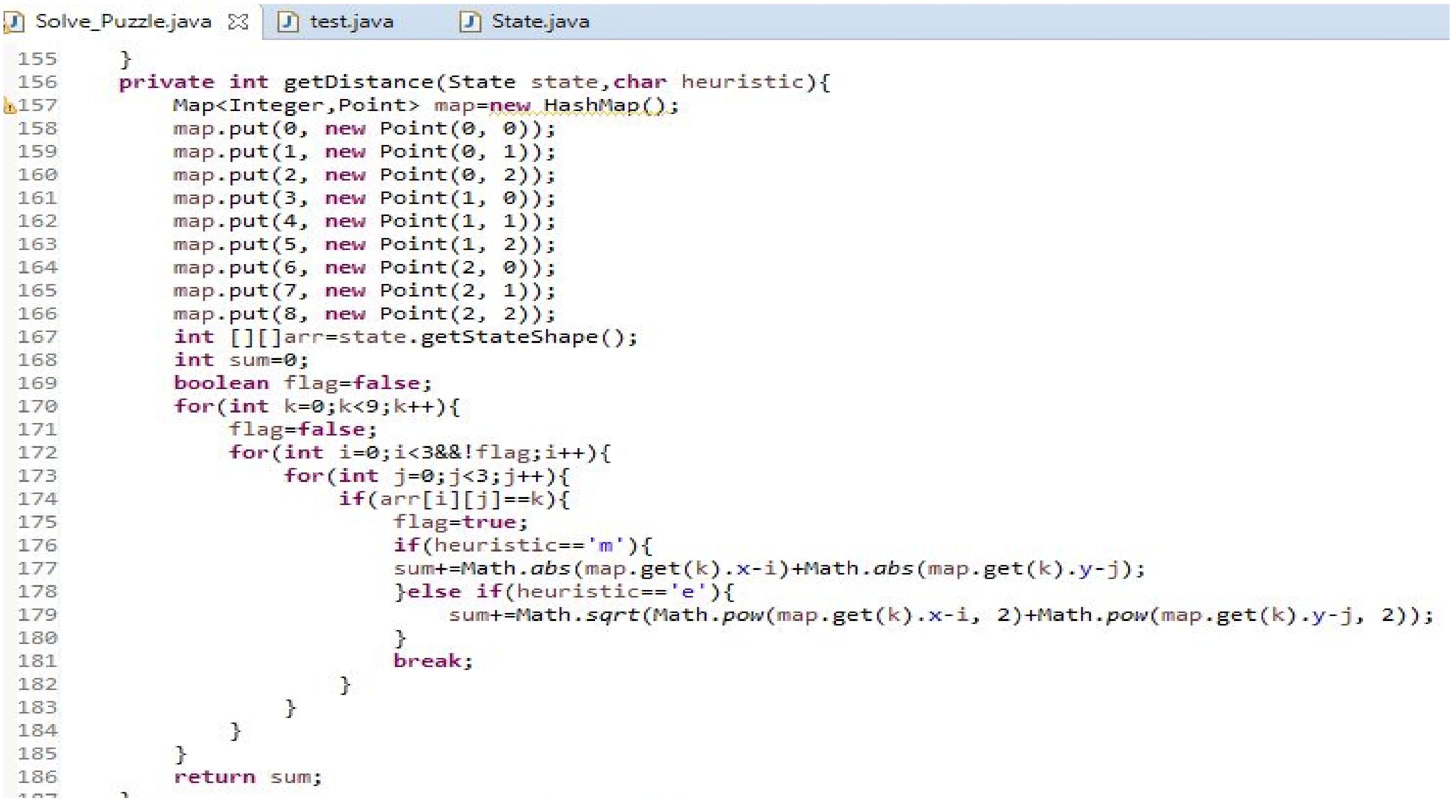
Main methods used :

1) getDistance :

Takes as parameters the state to calculate heuristic for and a character to determine what heuristic to use :

For heuristic= ‘m’ get manhatten distance

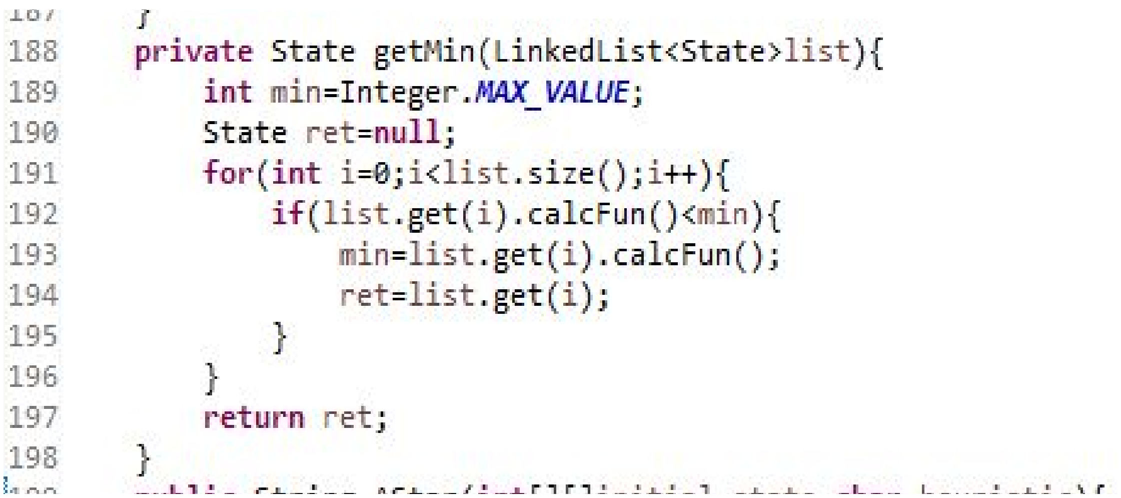
For heuristic =’e’ get euclidean distance



```java
    }
    private int getDistance(State state,char heuristic){
        Map<Integer,Point> map=new HashMap();
        map.put(0, new Point(0, 0));
        map.put(1, new Point(0, 1));
        map.put(2, new Point(0, 2));
        map.put(3, new Point(1, 0));
        map.put(4, new Point(1, 1));
        map.put(5, new Point(1, 2));
        map.put(6, new Point(2, 0));
        map.put(7, new Point(2, 1));
        map.put(8, new Point(2, 2));
        int [][]arr=state.getStateShape();
        int sum=0;
        boolean flag=false;
        for(int k=0;k<9;k++){
            flag=false;
            for(int i=0;i<3&&!flag;i++){
                for(int j=0;j<3;j++){
                    if(arr[i][j]==k){
                        flag=true;
                        if(heuristic=='m'){
                        sum+=Math.abs(map.get(k).x-i)+Math.abs(map.get(k).y-j);
                        }else if(heuristic=='e'){
                            sum+=Math.sqrt(Math.pow(map.get(k).x-i, 2)+Math.pow(map.get(k).y-j, 2));
                        }
                        break;
                    }
                }
            }
        }
        return sum;
```

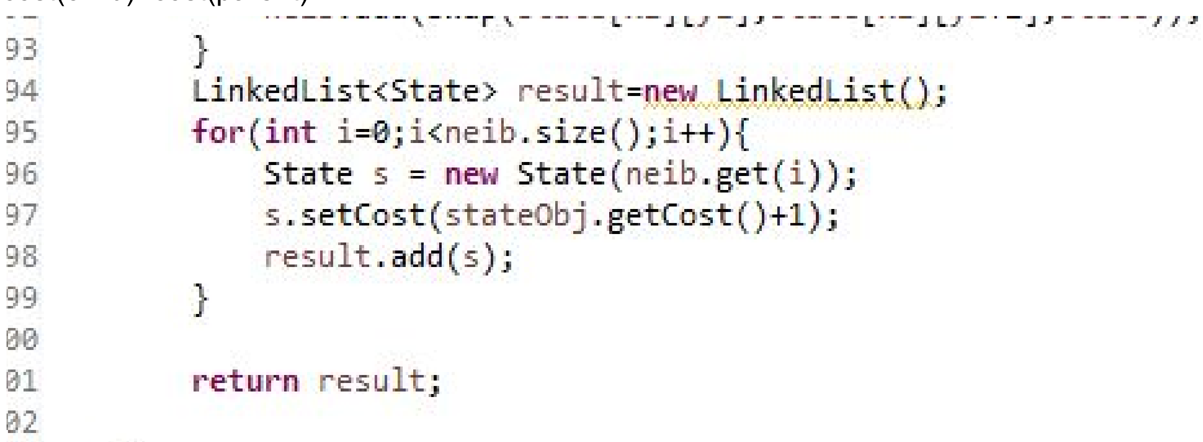
2)get Min

Takes the frontier list of states and returns the state with least f(x)=h(x)+cost(x)



```
    private State getMin(LinkedList<State>list){
        int min=Integer.MAX_VALUE;
        State ret=null;
        for(int i=0;i<list.size();i++){
            if(list.get(i).calcFun()<min){
                min=list.get(i).calcFun();
                ret=list.get(i);
            }
        }
        return ret;
    }
```

3)cost is calculated inside th function get neighbours as it propagates from parent to child cost(child)=cost(parent)+1

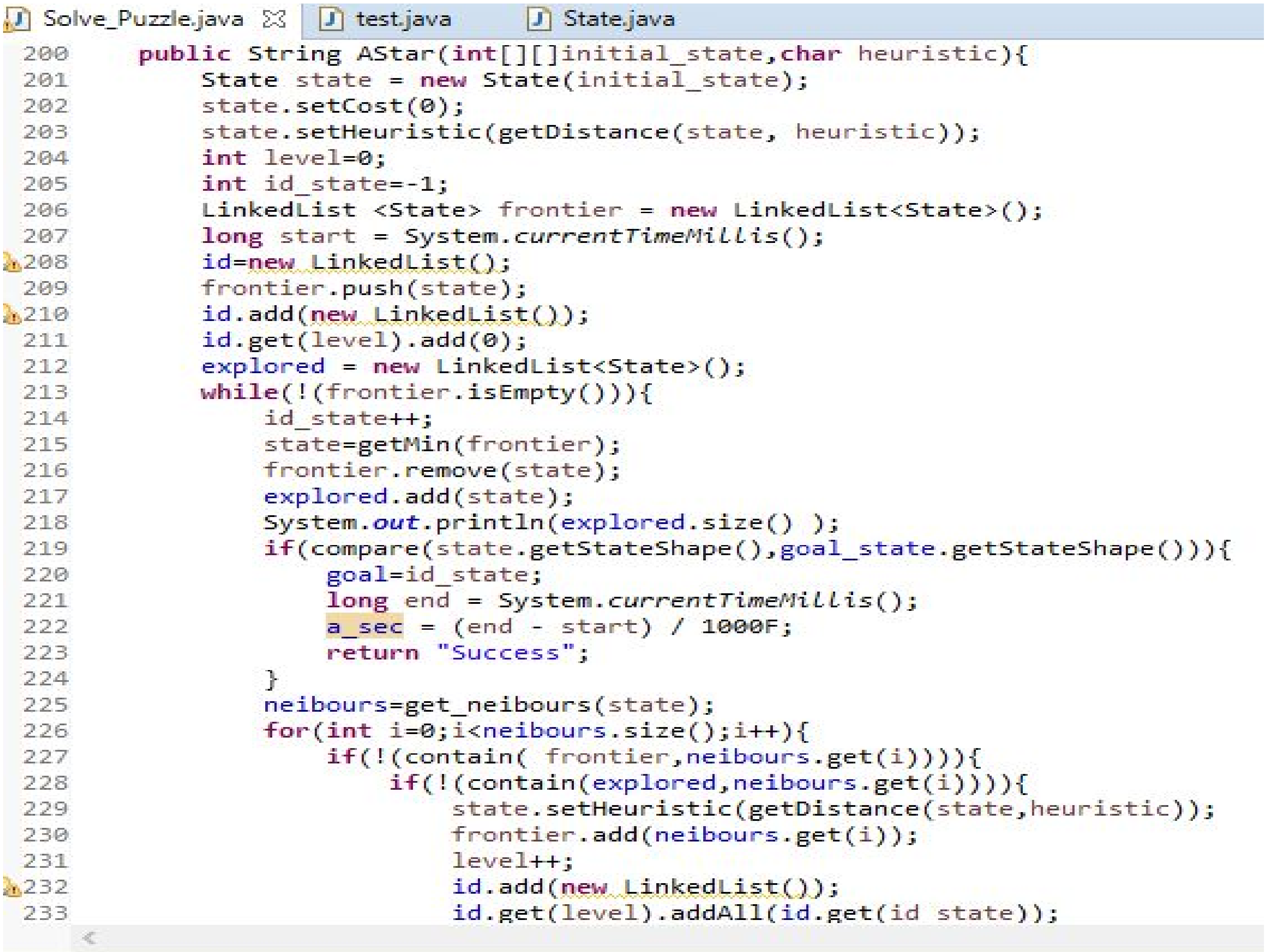


```
        }
        LinkedList<State> result=new LinkedList();
        for(int i=0;i<neib.size();i++){
            State s = new State(neib.get(i));
            s.setCost(stateObj.getCost()+1);
            result.add(s);
        }

        return result;

```

Function used for A* using previous methods is :



```
    public String AStar(int[][]initial_state,char heuristic){
        State state = new State(initial_state);
        state.setCost(0);
        state.setHeuristic(getDistance(state, heuristic));
        int level=0;
        int id_state=-1;
        LinkedList <State> frontier = new LinkedList<State>();
        long start = System.currentTimeMillis();
        id=new LinkedList();
        frontier.push(state);
        id.add(new LinkedList());
        id.get(level).add(0);
        explored = new LinkedList<State>();
        while(!(frontier.isEmpty())){
            id_state++;
            state=getMin(frontier);
            frontier.remove(state);
            explored.add(state);
            System.out.println(explored.size() );
            if(compare(state.getStateShape(),goal_state.getStateShape())){
                goal=id_state;
                long end = System.currentTimeMillis();
                a_sec = (end - start) / 1000F;
                return "Success";
            }
            neibours=get_neibours(state);
            for(int i=0;i<neibours.size();i++){
                if(!(contain( frontier,neibours.get(i)))){
                    if(!(contain(explored,neibours.get(i)))){
                        state.setHeuristic(getDistance(state,heuristic));
                        frontier.add(neibours.get(i));
                        level++;
                        id.add(new LinkedList());
                        id.get(level).addAll(id.get(id_state));
```