Tanner Juby
Programming Assignment 1
Problem Solving Using Searching

**PROBLEM:**

Solve a Sliding-Tile Puzzle using four different algorithms: A* using Heuristic 1, A* using Heuristic 2, Backtracking, and GraphSearch.

**CONCLUSION:**

Of all the algorithms, graph search proved to be the slowest most of the time. Both the A* algorithms were a little slower than graph search for the 1-move state, and only A* heuristic two was slower than graph search up to the 1-move state, but after that, graph search became very slow at finding the solution. The A* algorithm with heuristic 1 was faster than with heuristic 2 for the first few tests, but after the 5-move puzzle, heuristic 2 became much more efficient. Backtracking was in the middle of the A* algorithms and the Graph Search Algorithm in terms of efficiency.

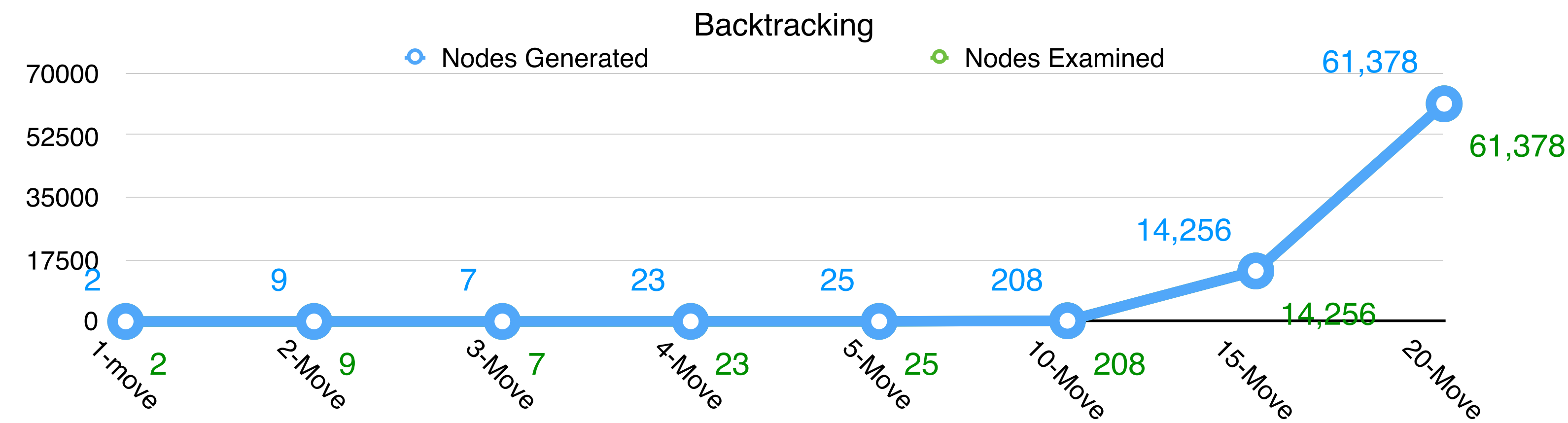
| Backtracking | Nodes Generated | Nodes Examined |
|---|---|---|
| Trivial | 1 | 1 |
| 1-move | 2 | 2 |
| 2-Move | 9 | 9 |
| 3-Move | 7 | 7 |
| 4-Move | 23 | 23 |
| 5-Move | 25 | 25 |
| 10-Move | 208 | 208 |
| 15-Move | 14256 | 14256 |
| 20-Move | 61378 | 61378 |

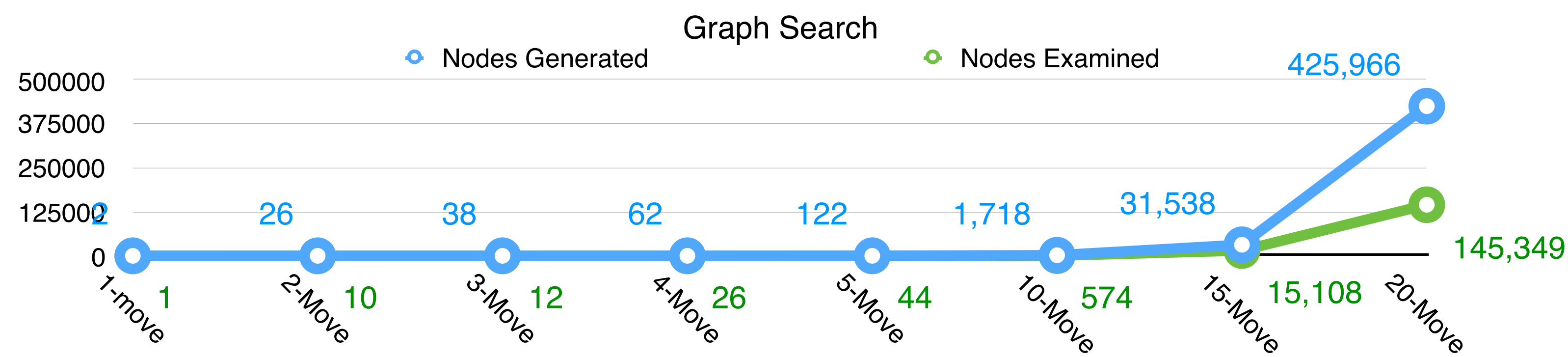
| Graph Search | Nodes Generated | Nodes Examined |
|---|---|---|
| Trivial | 1 | 1 |
| 1-move | 2 | 1 |
| 2-Move | 26 | 10 |
| 3-Move | 38 | 12 |
| 4-Move | 62 | 26 |
| 5-Move | 122 | 44 |
| 10-Move | 1718 | 574 |
| 15-Move | 31538 | 15108 |
| 20-Move | 425966 | 145349 |

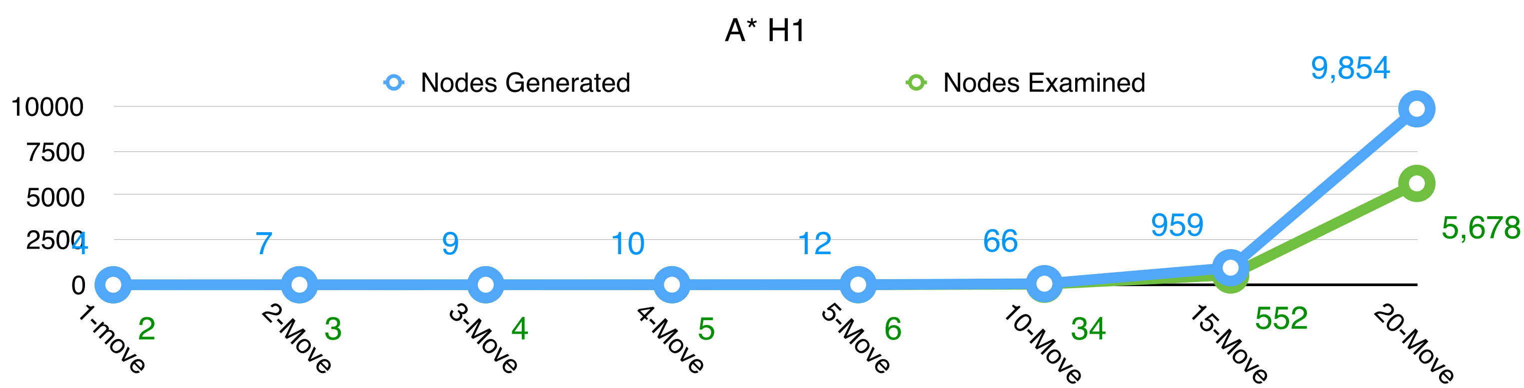
| A* H1 | Nodes Generated | Nodes Examined |
|---|---|---|
| Trivial | 1 | 1 |
| 1-move | 4 | 2 |
| 2-Move | 7 | 3 |
| 3-Move | 9 | 4 |
| 4-Move | 10 | 5 |
| 5-Move | 12 | 6 |
| 10-Move | 66 | 34 |
| 15-Move | 959 | 552 |
| 20-Move | 9854 | 5678 |

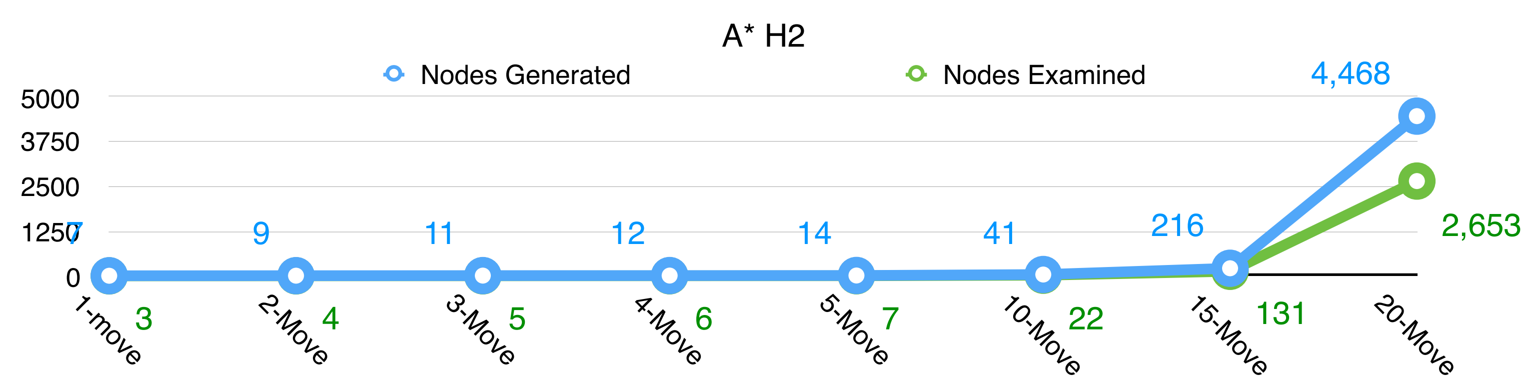
| A* H2 | Nodes Generated | Nodes Examined |
|---|---|---|
| Trivial | 1 | 1 |
| 1-move | 7 | 3 |
| 2-Move | 9 | 4 |
| 3-Move | 11 | 5 |
| 4-Move | 12 | 6 |
| 5-Move | 14 | 7 |
| 10-Move | 41 | 22 |
| 15-Move | 216 | 131 |
| 20-Move | 4468 | 2653 |

Tanner Juby
Programming Assignment 1
Problem Solving Using Searching



Tanner Juby
Programming Assignment 1
Problem Solving Using Searching

**RESULTS:**

--------------------

SOLUTIONS TO START STATE: trivial.json

Start State is Valid

START STATE
0 1 2
3 4 5
6 7 8

GOAL STATE
0 1 2
3 4 5
6 7 8

BACKTRACKING ALGORITHM SOLUTION:
Start State is the Goal State. No Solution
TOTAL MOVES: 0
TOTAL NODES GENERATED: 1
TOTAL STATES EXAMINED: 1

GRAPH SEARCH ALGORITHM SOLUTION:
Start State is the Goal State. No Solution
TOTAL MOVES: 0
TOTAL NODES GENERATED: 2
TOTAL STATES EXAMINED: 1

A STAR ALGORITHM WITH HEURISTIC ONE SOLUTION:
Start State is the Goal State. No Solution
TOTAL MOVES: 0
TOTAL NODES GENERATED: 1
TOTAL STATES EXAMINED: 1

A STAR ALGORITHM WITH HEURISTIC ONE SOLUTION:
Start State is the Goal State. No Solution
TOTAL MOVES: 0
TOTAL NODES GENERATED: 1
TOTAL STATES EXAMINED: 1

--------------------

--------------------

SOLUTIONS TO START STATE: 1-move.json

Start State is Valid

START STATE
3 1 2
0 4 5
6 7 8

GOAL STATE
0 1 2
3 4 5
6 7 8

BACKTRACKING ALGORITHM SOLUTION:
UP
TOTAL MOVES: 1

TOTAL NODES GENERATED: 2
TOTAL STATES EXAMINED: 2

GRAPH SEARCH ALGORITHM SOLUTION:
UP
TOTAL MOVES: 1
TOTAL NODES GENERATED: 8
TOTAL STATES EXAMINED: 2

A STAR ALGORITHM WITH HEURISTIC ONE SOLUTION:
UP
TOTAL MOVES: 1
TOTAL NODES GENERATED: 4
TOTAL STATES EXAMINED: 2

A STAR ALGORITHM WITH HEURISTIC TWO SOLUTION:
UP
TOTAL MOVES: 1
TOTAL NODES GENERATED: 7
TOTAL STATES EXAMINED: 3

--------------------

--------------------

SOLUTIONS TO START STATE: 2-moves.json

Start State is Valid

START STATE

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

GOAL STATE

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

BACKTRACKING ALGORITHM SOLUTION:
UP LEFT
TOTAL MOVES: 2
TOTAL NODES GENERATED: 9
TOTAL STATES EXAMINED: 9

GRAPH SEARCH ALGORITHM SOLUTION:
UP LEFT
TOTAL MOVES: 2
TOTAL NODES GENERATED: 26
TOTAL STATES EXAMINED: 10

A STAR ALGORITHM WITH HEURISTIC ONE SOLUTION:
UP LEFT
TOTAL MOVES: 2
TOTAL NODES GENERATED: 7
TOTAL STATES EXAMINED: 3

A STAR ALGORITHM WITH HEURISTIC TWO SOLUTION:
UP LEFT
TOTAL MOVES: 2
TOTAL NODES GENERATED: 9
TOTAL STATES EXAMINED: 4

--------------------

--------------------

Tanner Juby
Programming Assignment 1
Problem Solving Using Searching

SOLUTIONS TO START STATE: 3-moves.json

Start State is Valid

START STATE

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

GOAL STATE

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

BACKTRACKING ALGORITHM SOLUTION:
UP LEFT UP
TOTAL MOVES: 3
TOTAL NODES GENERATED: 7
TOTAL STATES EXAMINED: 7

GRAPH SEARCH ALGORITHM SOLUTION:
UP LEFT UP
TOTAL MOVES: 3
TOTAL NODES GENERATED: 38
TOTAL STATES EXAMINED: 12

A STAR ALGORITHM WITH HEURISTIC ONE SOLUTION:
UP LEFT UP
TOTAL MOVES: 3
TOTAL NODES GENERATED: 9
TOTAL STATES EXAMINED: 4

A STAR ALGORITHM WITH HEURISTIC TWO SOLUTION:
UP LEFT UP
TOTAL MOVES: 3
TOTAL NODES GENERATED: 11
TOTAL STATES EXAMINED: 5

--------------------

--------------------

SOLUTIONS TO START STATE: 4-moves.json

Start State is Valid

START STATE

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

GOAL STATE

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

BACKTRACKING ALGORITHM SOLUTION:
UP LEFT UP LEFT
TOTAL MOVES: 4
TOTAL NODES GENERATED: 23
TOTAL STATES EXAMINED: 23

GRAPH SEARCH ALGORITHM SOLUTION:
UP LEFT UP LEFT
TOTAL MOVES: 4

TOTAL NODES GENERATED: 62
TOTAL STATES EXAMINED: 26

A STAR ALGORITHM WITH HEURISTIC ONE SOLUTION:
UP LEFT UP LEFT
TOTAL MOVES: 4
TOTAL NODES GENERATED: 10
TOTAL STATES EXAMINED: 5

A STAR ALGORITHM WITH HEURISTIC TWO SOLUTION:
UP LEFT UP LEFT
TOTAL MOVES: 4
TOTAL NODES GENERATED: 12
TOTAL STATES EXAMINED: 6

--------------------

--------------------

SOLUTIONS TO START STATE: 5-moves.json

Start State is Valid

START STATE

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

GOAL STATE

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

BACKTRACKING ALGORITHM SOLUTION:
UP LEFT UP LEFT DOWN
TOTAL MOVES: 5
TOTAL NODES GENERATED: 25
TOTAL STATES EXAMINED: 25

GRAPH SEARCH ALGORITHM SOLUTION:
UP LEFT UP LEFT DOWN
TOTAL MOVES: 5
TOTAL NODES GENERATED: 122
TOTAL STATES EXAMINED: 44

A STAR ALGORITHM WITH HEURISTIC ONE SOLUTION:
UP LEFT UP LEFT DOWN
TOTAL MOVES: 5
TOTAL NODES GENERATED: 12
TOTAL STATES EXAMINED: 6

A STAR ALGORITHM WITH HEURISTIC TWO SOLUTION:
UP LEFT UP LEFT DOWN
TOTAL MOVES: 5
TOTAL NODES GENERATED: 14
TOTAL STATES EXAMINED: 7

--------------------

--------------------

SOLUTIONS TO START STATE: 10-moves.json

Start State is Valid

START STATE

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

GOAL STATE

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

BACKTRACKING ALGORITHM SOLUTION:
UP LEFT UP LEFT DOWN DOWN RIGHT UP RIGHT UP
TOTAL MOVES: 10
TOTAL NODES GENERATED: 208
TOTAL STATES EXAMINED: 208

GRAPH SEARCH ALGORITHM SOLUTION:
UP LEFT UP LEFT DOWN DOWN RIGHT UP RIGHT UP
TOTAL MOVES: 10
TOTAL NODES GENERATED: 1718
TOTAL STATES EXAMINED: 574

A STAR ALGORITHM WITH HEURISTIC ONE SOLUTION:
UP LEFT UP LEFT DOWN DOWN RIGHT UP RIGHT UP
TOTAL MOVES: 10
TOTAL NODES GENERATED: 66
TOTAL STATES EXAMINED: 34

A STAR ALGORITHM WITH HEURISTIC TWO SOLUTION:
UP LEFT UP LEFT DOWN DOWN RIGHT UP RIGHT UP
TOTAL MOVES: 10
TOTAL NODES GENERATED: 41
TOTAL STATES EXAMINED: 22

--------------------

--------------------

SOLUTIONS TO START STATE: 15-moves.json

Start State is Valid

START STATE

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

GOAL STATE

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

BACKTRACKING ALGORITHM SOLUTION:
UP LEFT UP LEFT DOWN DOWN RIGHT UP RIGHT UP LEFT DOWN LEFT UP RIGHT
TOTAL MOVES: 15
TOTAL NODES GENERATED: 14256
TOTAL STATES EXAMINED: 14256

GRAPH SEARCH ALGORITHM SOLUTION:
UP LEFT UP LEFT DOWN DOWN RIGHT UP RIGHT UP LEFT DOWN LEFT UP RIGHT
TOTAL MOVES: 15
TOTAL NODES GENERATED: 31538
TOTAL STATES EXAMINED: 15108

A STAR ALGORITHM WITH HEURISTIC ONE SOLUTION:
UP LEFT UP LEFT DOWN DOWN RIGHT UP RIGHT UP LEFT DOWN LEFT UP RIGHT
TOTAL MOVES: 15

TOTAL NODES GENERATED: 959
TOTAL STATES EXAMINED: 552

A STAR ALGORITHM WITH HEURISTIC TWO SOLUTION:
UP LEFT UP LEFT DOWN DOWN RIGHT UP RIGHT UP LEFT DOWN LEFT UP RIGHT
TOTAL MOVES: 15
TOTAL NODES GENERATED: 226
TOTAL STATES EXAMINED: 131

--------------------

--------------------

SOLUTIONS TO START STATE: 20-moves.json

Start State is Valid

START STATE

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

GOAL STATE

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

BACKTRACKING ALGORITHM SOLUTION:
LEFT LEFT UP RIGHT UP RIGHT DOWN LEFT UP LEFT DOWN RIGHT DOWN RIGHT UP UP LEFT DOWN LEFT DOWN
TOTAL MOVES: 20
TOTAL NODES GENERATED: 61378
TOTAL STATES EXAMINED: 61378

GRAPH SEARCH ALGORITHM SOLUTION:
LEFT LEFT UP RIGHT UP RIGHT DOWN LEFT UP LEFT DOWN RIGHT DOWN RIGHT UP UP LEFT DOWN LEFT DOWN
TOTAL MOVES: 20
TOTAL NODES GENERATED: 425966
TOTAL STATES EXAMINED: 145349

A STAR ALGORITHM WITH HEURISTIC ONE SOLUTION:
UP LEFT UP LEFT DOWN DOWN RIGHT UP RIGHT UP LEFT DOWN LEFT UP RIGHT DOWN RIGHT DOWN LEFT LEFT
TOTAL MOVES: 20
TOTAL NODES GENERATED: 9854
TOTAL STATES EXAMINED: 5678

A STAR ALGORITHM WITH HEURISTIC TWO SOLUTION:
UP LEFT UP LEFT DOWN DOWN RIGHT UP RIGHT UP LEFT DOWN LEFT UP RIGHT DOWN RIGHT DOWN LEFT LEFT
TOTAL MOVES: 20
TOTAL NODES GENERATED: 4468
TOTAL STATES EXAMINED: 2653

— — — — — — — — — —

Tanner Juby
Programming Assignment 1
Problem Solving Using Searching

**SOURCE CODE**

Files:

- main.swift
- PuzzleState.swift
- Algorithms.swift
- Queue.swift
- LinkedList.swift

**main.swift**

```
//
//  main.swift
//  SlidingPuzzle
//
//  Created by Tanner Juby on 2/12/17.
//  Copyright © 2017 Juby. All rights reserved.
//

import Foundation

let basePath = "/Users/TannerJuby/Dropbox/School/AI/PA1/PA1/"

var totalNodes = 0

var backtrackingNodesGenerated = 0
var graphSearchNodesGenerated = 0
var aStar1NodesGenerated = 0
var aStar2NodesGenerated = 0

var backtrackingStatesExamined = 0
var graphSearchStatesExamined = 0
var aStar1StatesExamined = 0
var aStar2StatesExamined = 0

// MARK - Retrieve the start states to be tested

/*
 Retrieve the Start State

 Gets start state from JSON file and creates the starting node.
 Calls the different algortihms to solve the puzzle
 */
let startStates = ["trivial.json", "1-move.json", "2-moves.json", "3-moves.json", "4-moves.json", "5-
moves.json", "10-moves.json", "15-moves.json", "20-moves.json", "25-moves.json", "15-
puzzle.json"]

for ss in startStates {

    let path = basePath + ss

    if let jsonData = NSData(contentsOfFile: path) {
        do {
            let object = try JSONSerialization.jsonObject(with: jsonData as Data,
options: .allowFragments)
            if let dictionary = object as? [String: AnyObject] {

                totalNodes = 0
                let startState = PuzzleState(dictionary: dictionary)

                backtrackingNodesGenerated = 0
                graphSearchNodesGenerated = 0
                aStar1NodesGenerated = 1
                aStar2NodesGenerated = 1
```

```
            backtrackingStatesExamined = 0
            graphSearchStatesExamined = 0
            aStar1StatesExamined = 0
            aStar2StatesExamined = 0

            print("--------------------\n")

            if startState.isValid() {
                print("SOLUTIONS TO START STATE: \(ss)\n")

                print("Start State is Valid\n")

                print("START STATE")
                startState.printState()

                print("GOAL STATE")
                startState.printGoal(goal: dictionary["goal"] as! [[Int]])

                var algorithms = Algorithms.init()

                // Run the A* Heuristic 1 Algorithm
                let AStarH1Solution : [String] = algorithms.algoAStarH1(start: startState, goal:
dictionary["goal"] as! [[Int]])

                // Run the A* Heuristic 2 Algorithm
                totalNodes = 0
                let AStarH2Solution : [String] = algorithms.algoAStarH2(start: startState, goal:
dictionary["goal"] as! [[Int]])

                // Run the Backtracking Algorithm
                totalNodes = 0

                let treeRootBacktrack = algorithms.buildTree(root: startState, currentDepth: 0,
maxDepth: AStarH2Solution.count)

                let backtrackSolution : [String] = algorithms.startbacktracking(node:
treeRootBacktrack, goal: dictionary["goal"] as! [[Int]])

                // Run the Graph Search Algorithm
                totalNodes = 0
                let treeRootGraph = algorithms.buildTree(root: startState, currentDepth: 0, maxDepth:
AStarH2Solution.count)

                let graphSearchSolution : [String] = algorithms.graphSearch(root: treeRootGraph,
goal: dictionary["goal"] as! [[Int]])

                // Print the results
                algorithms.printBacktrackingResult(solution: backtrackSolution)
                algorithms.printGraphSearchResult(solution: graphSearchSolution)
                algorithms.printA1Result(solution: AStarH1Solution)
                algorithms.printA2Result(solution: AStarH2Solution)

                print("--------------------\n")


            } else {
                print("ERROR: Start State is invalid")
            }
        }
    } catch {
        print("ERROR: Could not serialize the data from file")
    }
} else {
    print("ERROR: Could not get data from file")
}
}
```

Tanner Juby
Programming Assignment 1
Problem Solving Using Searching

**PuzzleState.swift**

```
//
//  PuzzleState.swift
//  SlidingPuzzle
//
//  Created by Tanner Juby on 2/12/17.
//  Copyright © 2017 Juby. All rights reserved.
//

import Foundation

enum Move {
    case None

    case Up
    case Down
    case Left
    case Right
}

struct EmptyLocation {
    var row: Int
    var column: Int
}


class PuzzleState {


    // MARK: - Class Variables

    var key : Int
    var move : Move

    var n : Int
    var startState : [[Int]]
    var emptyLocation : EmptyLocation!

    var possibleMoves : [PuzzleState] = []
    weak var previousState: PuzzleState?


    // MARK: - Class Initializers

    init(dictionary: Dictionary<String, AnyObject>) {

        key = totalNodes
        move = .None

        n = dictionary["n"] as! Int
        startState = dictionary["start"] as! [[Int]]

        setEmptyTile()

        previousState = nil
    }

    init(n: Int, startState: [[Int]], previousState: PuzzleState, move: Move) {

        totalNodes += 1
        self.key = totalNodes
        self.move = move

        self.n = n
```

```
    self.startState = startState

    setEmptyTile()

    self.previousState = previousState
}


// MARK: - Class Functions

/**
 Print Board

 Prints out the current state of the puzzle board
 */
func printState() {

    var printObject = ""

    for i in 0 ..< n {
        for j in 0 ..< n {
            printObject += "\(startState[i][j]) \t"
        }
        printObject += "\n"
    }

    print("\(printObject)")
}

/**
 Print Goal

 Prints out the goal state
*/
func printGoal(goal: [[Int]]) {

    var printObject = ""

    for i in 0 ..< n {
        for j in 0 ..< n {
            printObject += "\(goal[i][j]) \t"
        }
        printObject += "\n"
    }

    print("\(printObject)")
}


/**
 Find Empty Tile

 Finds the empty location in the current state
 */
func setEmptyTile()  {

    for i in 0 ..< n {
        for j in 0 ..< n {
            if startState[i][j] == 0 {
                emptyLocation = EmptyLocation(row: i, column: j)
            }
        }
    }
}
```

```
    /**
     Set Children
     
     Sets up the children of the node
    */
    func setMoves() {
        
        possibleMoves.removeAll()
        
        // Test Up
        if emptyLocation.row != 0 && move != .Down {
            
            var newState = startState
            
            // MAKE SWAP
            newState[emptyLocation.row][emptyLocation.column] = startState[emptyLocation.row-1]
[emptyLocation.column]
            newState[emptyLocation.row-1][emptyLocation.column] = startState[emptyLocation.row]
[emptyLocation.column]
            
            let tempChild = PuzzleState(n: n, startState: newState, previousState: self, move: .Up)
            
            possibleMoves.append(tempChild)
        }
        
        // Test Down
        if emptyLocation.row != n-1 && move != .Up {
            
            var newState = startState
            
            // MAKE SWAP
            newState[emptyLocation.row][emptyLocation.column] = startState[emptyLocation.row+1]
[emptyLocation.column]
            newState[emptyLocation.row+1][emptyLocation.column] = startState[emptyLocation.row]
[emptyLocation.column]
            
            let tempChild = PuzzleState(n: n, startState: newState, previousState: self, move: .Down)
            
            possibleMoves.append(tempChild)
        }
        
        // Test Left
        if emptyLocation.column != 0 && move != .Right {
            
            var newState = startState
            
            // MAKE SWAP
            newState[emptyLocation.row][emptyLocation.column] = startState[emptyLocation.row]
[emptyLocation.column-1]
            newState[emptyLocation.row][emptyLocation.column-1] = startState[emptyLocation.row]
[emptyLocation.column]
            
            let tempChild = PuzzleState(n: n, startState: newState, previousState: self, move: .Left)
            
            possibleMoves.append(tempChild)
        }
        
        // Test Right
        if emptyLocation.column != n-1  && move != .Left {
            
            var newState = startState
            
            // MAKE SWAP
            newState[emptyLocation.row][emptyLocation.column] = startState[emptyLocation.row]
[emptyLocation.column+1]
```

```
            newState[emptyLocation.row][emptyLocation.column+1] = startState[emptyLocation.row]
[emptyLocation.column]

            let tempChild = PuzzleState(n: n, startState: newState, previousState: self, move: .Right)

            possibleMoves.append(tempChild)
        }
    }

    /*
     Is Valid

     Checks to ensure the current state of the puzzle is valid

     Return - Bool
     */
    func isValid() -> Bool {

        let total = n*n

        var checkArray = [Int]()

        for i in 0 ..< total {
            checkArray.append(i)
        }

        for k in 0 ..< checkArray.count {

            var doesExist = false

            for i in 0 ..< n {
                for j in 0 ..< n {
                    if startState[i][j] == checkArray[k] {
                        doesExist = true
                    }
                }
            }

            if doesExist == false {
                return false
            }
        }
        return true
    }

    /**
     Compare with State

     Compares class's start state with another state
    */
    func compareWith(state: PuzzleState) -> Bool {

        for i in 0 ..< n {
            for j in 0 ..< n {
                if startState[i][j] == state.startState[i][j] {
                    return false
                }
            }
        }
        return true

    }

    /**
     Is Goal
```

```
     Compares class's start state with another state
     */
    func isGoal(goal: [[Int]]) -> Bool {

        for i in 0 ..< n {
            for j in 0 ..< n {
                if startState[i][j] != goal[i][j] {
                    return false
                }
            }
        }
        return true

    }

}
```

**Algorithms.swift**

```
//
//  Algorithms.swift
//  SlidingPuzzle
//
//  Created by Tanner Juby on 2/12/17.
//  Copyright © 2017 Juby. All rights reserved.
//

import Foundation


/**
 Algorithms

 The Class of Algorithms
 */
class Algorithms {

    /**
     Algorithm A*, Hueristic 1

     Performs the A* Algorithm with Hueristic 1
    */
    func algoAStarH1(start: PuzzleState, goal: [[Int]]) -> [String] {

        var openSet : [Int] = [start.key]
        var closedSet : [Int] = []

        var states : Dictionary<Int, PuzzleState> = Dictionary()
        states[start.key] = start

        var gScore : Dictionary<Int, Int> = Dictionary()
        gScore[start.key] = 0

        var hScore : Dictionary<Int, Int> = Dictionary()
        hScore[start.key] = heuristicOne(state: start, goalState: goal)

        var fScore : Dictionary<Int, Int> = Dictionary()
        fScore[start.key] = hScore[start.key]

        while !openSet.isEmpty {
            aStar1StatesExamined += 1
            // Find lowest score in fScore
            let xKey = findFMin(fScore: fScore, openSet: openSet)!
```

```
        let x = states[xKey]!

        // Check if X is equal to the goal
        if (x.isGoal(goal: goal)) {

            return resconstructPath(state: x)
        }

        let openIndex = findIndexOf(array: openSet, value: (x.key))

        openSet.remove(at: openIndex!)
        closedSet.append((x.key))

        x.setMoves()
        aStar1NodesGenerated += x.possibleMoves.count

        for move in (x.possibleMoves) {
            states[move.key] = move

            if findIndexOf(array: closedSet, value: move.key) != nil {
                continue
            }

            let tentativeGScore = resconstructPath(state: move).count

            if findIndexOf(array: openSet, value: move.key) == nil {
                openSet.append(move.key)
            } else if tentativeGScore >= gScore[move.key]! {
                continue
            }

            gScore[move.key] = tentativeGScore
            hScore[move.key] = heuristicOne(state: move, goalState: goal)
            fScore[move.key] = gScore[move.key]! + hScore[move.key]!

        }
    }
    return ["ERROR: Could Not Find Optimal Solution"]
}


/**
 Algorithm A*, Hueristic 2

 Performs the A* Algorithm with Hueristic 2
 */
func algoAStarH2(start: PuzzleState, goal: [[Int]]) -> [String] {

    var openSet : [Int] = [start.key]
    var closedSet : [Int] = []

    var states : Dictionary<Int, PuzzleState> = Dictionary()
    states[start.key] = start

    var gScore : Dictionary<Int, Int> = Dictionary()
    gScore[start.key] = 0

    var hScore : Dictionary<Int, Int> = Dictionary()
    hScore[start.key] = heuristicTwo(state: start, goalState: goal)

    var fScore : Dictionary<Int, Int> = Dictionary()
    fScore[start.key] = hScore[start.key]

    while !openSet.isEmpty {
        aStar2StatesExamined += 1
        // Find lowest score in fScore
```

```
        let xKey = findFMin(fScore: fScore, openSet: openSet)!
        let x = states[xKey]!

        // Check if X is equal to the goal
        if (x.isGoal(goal: goal)) {
            return resconstructPath(state: x)
        }

        let openIndex = findIndexOf(array: openSet, value: (x.key))

        openSet.remove(at: openIndex!)
        closedSet.append((x.key))

        x.setMoves()
        aStar2NodesGenerated += x.possibleMoves.count

        for move in (x.possibleMoves) {
            states[move.key] = move

            if findIndexOf(array: closedSet, value: move.key) != nil {
                continue
            }

            let tentativeGScore = resconstructPath(state: move).count

            if findIndexOf(array: openSet, value: move.key) == nil {
                openSet.append(move.key)
            } else if tentativeGScore >= gScore[move.key]! {
                continue
            }

            gScore[move.key] = tentativeGScore
            hScore[move.key] = heuristicTwo(state: move, goalState: goal)
            fScore[move.key] = gScore[move.key]! + hScore[move.key]!
        }
    }
    return ["ERROR: Could Not Find Optimal Solution"]
}

/**
 Graph Search

 Implementation of the graph search algorithm
 */
func graphSearch(root: PuzzleState, goal: [[Int]]) -> [String] {

    var openSet : [Int] = []
    var queue = Queue<Int>()

    var states : Dictionary<Int, PuzzleState> = Dictionary()
    states[root.key] = root

    queue.enqueue(root.key)

    while !queue.isEmpty {
        graphSearchStatesExamined += 1
        let xKey = queue.dequeue()!
        let x = states[xKey]!

        if (x.isGoal(goal: goal)) {
            return resconstructPath(state: x)
        }

        for move in x.possibleMoves {
            if findIndexOf(array: openSet, value: move.key) == nil {
                openSet.append(move.key)
```

```
                queue.enqueue(move.key)
                states[move.key] = move
            }
        }
    }

    return ["ERROR: Could Not Find Optimal Solution"]
}

// Variables for backtracking algorithm
var backtrackSolution = 0
var moves : [Int : PuzzleState] = [:]

/**
 Start Backtracking

 Calls backtracking algorithm and handles the solution
*/
func startbacktracking(node: PuzzleState, goal: [[Int]]) -> [String] {

    if backtracking(node: node, goal: goal) {
        let solution = moves[backtrackSolution]
        return resconstructPath(state: solution!)
    } else {
        return ["ERROR: Could Not Find Optimal Solution"]
    }
}

/**
 Backtracking

 The backtracking algorithm
*/
func backtracking(node: PuzzleState, goal: [[Int]]) -> Bool {
    backtrackingStatesExamined += 1
    backtrackingNodesGenerated += 1

    moves[node.key] = node

    if node.possibleMoves.count == 0 {
        if node.isGoal(goal: goal) {
            backtrackSolution = node.key
            return true
        } else {
            return false
        }
    } else {
        for move in node.possibleMoves {
            if backtracking(node: move, goal: goal) {
                return true
            }
        }
        return false
    }
}
}


/**
 A* Algorithms Extension

 Helper Functions for the A* Algorithms
 */
extension Algorithms {

    /**
```

```
 Reconstruct Path

 Gets the path of state to start state
*/
func resconstructPath(state: PuzzleState) -> [String] {

    var returnArray : [String] = []

    var tempState = state

    while tempState.previousState != nil {

        switch tempState.move {
        case .Up:
            returnArray.append("UP")
        case .Down:
            returnArray.append("DOWN")
        case .Right:
            returnArray.append("RIGHT")
        case .Left:
            returnArray.append("LEFT")
        case.None:
            returnArray.append("COMPLETE")
        }

        if tempState.previousState != nil {
            tempState = tempState.previousState!
        } else {
            return returnArray
        }
    }
    return returnArray
}

/**
 Heuristic One

 Counts the number of out of place tiles
*/
func heuristicOne(state: PuzzleState, goalState: [[Int]]) -> Int {
    var outOfPlaceCount = 0

    for i in 0 ..< state.n {
        for j in 0 ..< state.n {
            if goalState[i][j] != state.startState[i][j] && goalState[i][j] != 0 && state.startState[i][j] != 0 {
                outOfPlaceCount += 1
            }
        }
    }

    return outOfPlaceCount
}

/**
 Heuristic Two

 Counts the Manhatten Distance of the graph
 */
func heuristicTwo(state: PuzzleState, goalState: [[Int]]) -> Int {

    var distance = 0

    for i in 0 ..< state.n {
        for j in 0 ..< state.n {
            let value = state.startState[i][j]
            if goalState[i][j] != state.startState[i][j] && goalState[i][j] != 0 && state.startState[i][j] != 0 {
```

```
                let targetX = (value - 1) / state.n
                let targetY = (value - 1) % state.n
                let dx = i - targetX
                let dy = j - targetY
                distance += abs(dx) + abs(dy)
            }
        }
    }

    return distance
}

/**
 Find FScore Min Va;ue

 Finds the node that has the lowest F Score
*/
func findFMin(fScore: Dictionary<Int, Int>, openSet: [Int]) -> Int? {

    var openFScore : Dictionary<Int, Int> = Dictionary()

    for key in openSet {
        openFScore[key] = fScore[key]
    }

    for (key, value) in openFScore {
        if value == openFScore.values.min() {
            return key
        }
    }

    return nil
}

/**
 Find Array Element

 Find the index of an array entry
*/
func findIndexOf(array: [Int], value: Int) -> Int? {

    for i in 0 ..< array.count {
        if array[i] == value {
            return i
        }
    }
    return nil
}
}


/**
 Backtracking Algorithm Extension

 Helper functions for the Backtracking algorithm
 */

var totalCount = 0

extension Algorithms {

    func buildTree(root: PuzzleState, currentDepth: Int, maxDepth: Int) -> PuzzleState {
        totalCount += 1
        graphSearchNodesGenerated += 1
```

```
        if currentDepth == maxDepth {
            return root
        } else {
            root.setMoves()

            var newMoves : [PuzzleState] = []

            for move in root.possibleMoves {
                let newMove = buildTree(root: move, currentDepth: currentDepth+1, maxDepth:
maxDepth)
                newMoves.append(newMove)
            }
            root.possibleMoves = newMoves
            return root
        }
    }


}


/**
 Print Results Extension

 Helper Function for printing out results
 */
extension Algorithms {

    func printA1Result(solution: [String]) {

        if solution.count == 0 {

            print("A STAR ALGORITHM WITH HEURISTIC ONE SOLUTION: ")
            print("Start State is the Goal State. No Solution")
            print("TOTAL MOVES: \(solution.count)")
            print("TOTAL NODES GENERATED: \(aStar1NodesGenerated)")
            print("TOTAL STATES EXAMINED: \(aStar1StatesExamined)")
            print()

        } else {

            var AStar1Print = ""
            for move in solution {
                AStar1Print += "\(move) "
            }

            print("A STAR ALGORITHM WITH HEURISTIC ONE SOLUTION: ")
            print(AStar1Print)
            print("TOTAL MOVES: \(solution.count)")
            print("TOTAL NODES GENERATED: \(aStar1NodesGenerated)")
            print("TOTAL STATES EXAMINED: \(aStar1StatesExamined)")
            print()
        }
    }

    func printA2Result(solution: [String]) {

        if solution.count == 0 {

            print("A STAR ALGORITHM WITH HEURISTIC ONE SOLUTION: ")
            print("Start State is the Goal State. No Solution")
            print("TOTAL MOVES: \(solution.count)")
            print("TOTAL NODES GENERATED: \(aStar2NodesGenerated)")
            print("TOTAL STATES EXAMINED: \(aStar2StatesExamined)")
            print()
```

```
    } else {

        var AStar2Print = ""
        for move in solution {
            AStar2Print += "\(move) "
        }

        print("A STAR ALGORITHM WITH HEURISTIC TWO SOLUTION: ")
        print(AStar2Print)
        print("TOTAL MOVES: \(solution.count)")
        print("TOTAL NODES GENERATED: \(aStar2NodesGenerated)")
        print("TOTAL STATES EXAMINED: \(aStar2StatesExamined)")
        print()
    }
}

func printBacktrackingResult(solution: [String]) {

    if solution.count == 0 {

        print("BACKTRACKING ALGORITHM SOLUTION: ")
        print("Start State is the Goal State. No Solution")
        print("TOTAL MOVES: \(solution.count)")
        print("TOTAL NODES GENERATED: \(backtrackingNodesGenerated)")
        print("TOTAL STATES EXAMINED: \(backtrackingStatesExamined)")
        print()

    } else {

        var backtrackingPrint = ""
        for move in solution {
            backtrackingPrint += "\(move) "
        }

        print("BACKTRACKING ALGORITHM SOLUTION: ")
        print(backtrackingPrint)
        print("TOTAL MOVES: \(solution.count)")
        print("TOTAL NODES GENERATED: \(backtrackingNodesGenerated)")
        print("TOTAL STATES EXAMINED: \(backtrackingStatesExamined)")
        print()
    }
}

func printGraphSearchResult(solution: [String]) {

    if solution.count == 0 {

        print("GRAPH SEARCH ALGORITHM SOLUTION: ")
        print("Start State is the Goal State. No Solution")
        print("TOTAL MOVES: \(solution.count)")
        print("TOTAL NODES GENERATED: \(graphSearchNodesGenerated)")
        print("TOTAL STATES EXAMINED: \(graphSearchStatesExamined)")
        print()

    } else {

        var graphSearchPrint = ""
        for move in solution {
            graphSearchPrint += "\(move) "
        }

        print("GRAPH SEARCH ALGORITHM SOLUTION: ")
        print(graphSearchPrint)
        print("TOTAL MOVES: \(solution.count)")
        print("TOTAL NODES GENERATED: \(graphSearchNodesGenerated)")
        print("TOTAL STATES EXAMINED: \(graphSearchStatesExamined)")
```

```
            print()
        }
    }

}
```

**Queue.swift**

```
 import Foundation

public struct Queue<T> {

    fileprivate var list = LinkedList<T>()

    public var isEmpty: Bool {
        return list.isEmpty
    }

    public mutating func enqueue(_ element: T) {
        list.append(element)
    }

    public mutating func dequeue() -> T? {
        guard !list.isEmpty, let element = list.first else {
            return nil
        }

        _ = list.remove(element)

        return element.value
    }

    public func peek() -> T? {
        return list.first?.value
    }
}

extension Queue: CustomStringConvertible {

    public var description: String {
        return list.description
    }
}
```

**LinkedList.swift**

```
public struct LinkedList<T>: CustomStringConvertible {

    private var head: Node<T>?
    private var tail: Node<T>?

    public init() { }

    public var isEmpty: Bool {
        return head == nil
    }
```

```
    public var first: Node<T>? {
        return head
    }

    public mutating func append(_ value: T) {
        let newNode = Node(value: value)
        if let tailNode = tail {
            newNode.previous = tailNode
            tailNode.next = newNode
        } else {
            head = newNode
        }
        tail = newNode
    }

    public mutating func remove(_ node: Node<T>) -> T {
        let prev = node.previous
        let next = node.next

        if let prev = prev {
            prev.next = next
        } else {
            head = next
        }
        next?.previous = prev

        if next == nil {
            tail = prev
        }

        node.previous = nil
        node.next = nil

        return node.value
    }

    public var description: String {
        var text = "["
        var node = head

        while node != nil {
            text += "\(node!.value)"
            node = node!.next
            if node != nil { text += ", " }
        }
        return text + "]"
    }
}

public class Node<T> {
    public var value: T
    public var next: Node<T>?
    public var previous: Node<T>?

    public init(value: T) {
        self.value = value
    }
}
```