File for the algorithms

Luke Garrigan

registration: 100086495

1 Introduction

Algorithm 1 DFS

```
1: visited ← emptySet
2: procedure DFS(state)
3:     if state = goal then
4:         return state
5:     for neighbour in neighbours(state) do
6:         if neighbour not in visited then
7:             DFS(neighbour)
8:     visited.add(state)
```

Algorithm 2 A*

```
1:  g(state)                                          ▷ The cost to reach the current state
2:  h(state)                          ▷ Estimated cost of the cheapest path from state to goal
3:  f(state) ← g(state) + h(state)
4:  neighbours(state)   ▷ Expands possible moves from current state ordered by g + h
5:  procedure ASTAR(startState)
6:      closedSet ← empty set
7:      openSet ← empty set
8:      openSet.add(startState)
9:      while openSet is not empty do
10:         currentState ← null
11:         for state in openSet do
12:             if f(state) < f(currentState) then
13:                 currentState ← state
14:         openSet.remove(currentState)
15:         closedSet.add(currentState)
16:         if currentState = goal then
17:             return currentState
18:         for neighbour in neighbours(currentState) do
19:             if neighbour not in closedSet then
20:                 if neighbour in openSet then
21:                     if g(currentState) < g(neighbour) then
22:                         g(neighbour) ← g(currentState)
23:                 else
24:                     openSet.add(neighbour)
```

Algorithm 3 Manhattan Distance

```
1:  procedure MANDIST(state)                          ▷ The current puzzle configuration
2:      total ← 0
3:      puzzleLength ← state.size()
4:      dimensions ← √puzzleLength
5:      for i ← 1, puzzleLength do                    ▷ Loops through each tile of the puzzle
6:          tileValue ← state[i]
7:          expectedRow ← (tileValue − 1) ÷ dimensions
8:          expectedCol ← (tileValue − 1)  mod dimensions
9:          rowNum ← i ÷ dimensions
10:         rowNum ← i mod dimensions
11:         total ← total + | expectedRow − rowNum | + | expectedCol − colNum |
12:     return total                                  ▷ The heuristic is the total
```

Algorithm 4 Iterative Deepening A Star

```
1: state                                          ▷ The current puzzle configuration
2: g(state)                                       ▷ The cost to reach the current state
3: h(state)                  ▷ Estimated cost of the cheapest path from state to goal
4: f(state) ← g(state) + h(state)
5: neighbours(state)   ▷ Expands possible moves from current state ordered by g + h

6: procedure IDASTAR(state)
7:     bound ← f(state)
8:     while not solved do                        ▷ Loops until a solution is found
9:         bound ← DFS(state, bound)          ▷ Performs a bounded depth-first search

10: procedure DFS(state, bound)
11:     if f(state) > bound then
12:         return f(state)
13:     if h(state) = 0 then                ▷ No more moves needed to reach goal state
14:         return solved
15:     min ← ∞
16:     for neighbour in neighbours(state) do
17:         temp ← DFS(neighbour, bound)
18:         if temp < min then
19:             min ← temp
20:     return min                          ▷ Returns the smallest of the neighbours
```

Algorithm 5 Breadth-First Search

```
1: procedure BFS(state)
2:     s ← empty set
3:     q ← empty queue
4:     q.add(state)
5:     q.enqueue(state)
6:     while q.size() > 0 do
7:         currentState ← q.dequeue()
8:         if currentState = goal then
9:             return current
10:        for neighbour in neighbours(currentState) do
11:            if neighbour is not in s then
12:                s.add(neighbour)
13:                q.enqueue(neighbour)
```

Algorithm 6 Is Current State Solvable

```
1:  procedure ISSOLVABLE(state)
2:      puzzleLength ← state.size()
3:      gridWidth ← √puzzleLength
4:      blankRowEven ← true
5:      for i ← 1, puzzleLength do
6:          if state[i] = 0 then
7:              blankRowEven ← (i/gridWidth) mod 2 = 0
8:              continue
9:              for j ← i+1, puzzleLength do
10:                 if state[i] > state[j] and state[j] ≠ 0 then
11:                     parity ←!parity
12:     if gridWidth mod 2 = 0 and blankRowEven then
13:         return !parity
14:     return parity
```

Algorithm 7 Iterative Deepening Depth-First Search

```
1:  goal                                   ▷ the completed goal state of the puzzle

2:  procedure IDDFS(state)
3:      for depth ← 0, ∞ do
4:          found ← DLS(state, depth)
5:          if found ≠ null then
6:              return found
7:  procedure DLS(state, depth)
8:      if depth = 0 and state = goal then
9:          return found
10:     if depth > 0 then
11:         for neighbour in neighbours(state) do
12:             found ← DLS(neighbour, depth − 1)
13:             if found ≠ null then
14:                 returnfound
15:     return null
```

Progress report

Description of project: aims, motivation	First	2.1	2.2	3	Fail
Description and understanding of issues and problems addressed in the project	First	2.1	2.2	3	Fail
Achievement so far according to what is reasonably expected for the type of project	First	2.1	2.2	3	Fail
Discussion and justification of changes to project aims, scope, workplan	First	2.1	2.2	3	Fail

Quality of writing

Clarity, structure correctness of writing	First	2.1	2.2	3	Fail

Comments

Supervisor: supervisor

Markers should circle the appropriate level of performance in each section. Report and evaluation sheet should be collected by the student from the supervisor.