File for the algorithms

Luke Garrigan

registration: 100086495

1 Introduction

Algorithm 1 DFS

```
visited ← emptySet
procedure DFS(state)
    if state = goal then
        return state
    for neighbour in neighbours(state) do
        if neighbour not in visited then
            DFS(neighbour)
    visited.add(state)
```

Algorithm 2 DFS

```
visited ← emptySet
procedure DFS(state)
    if state = goal then
        return state
    for neighbour in neighbours(state) do
        if neighbour not in visited then
            DFS(neighbour)
    visited.add(state)
```

Algorithm 3 A*

```
g(state)                                        ▷ The cost to reach the current state
h(state)                    ▷ Estimated cost of the cheapest path from state to goal
f(state) ← g(state) + h(state)
neighbours(state)    ▷ Expands possible moves from current state ordered by g + h
procedure ASTAR(startState)
    closedSet ← empty set
    openSet ← empty set
    openSet.add(startState)
    while openSet is not empty do
        currentState ← null
        for state in openSet do
            if f(state) < f(currentState) then
                currentState ← state
        openSet.remove(currentState)
        closedSet.add(currentState)
        if currentState = goal then
            return currentState
        for neighbour in neighbours(currentState) do
            if neighbour not in closedSet then
                if neighbour in openSet then
                    if g(currentState) < g(neighbour) then
                        g(neighbour) ← g(currentState)
                else
                    openSet.add(neighbour)
```

Algorithm 4 Manhattan Distance

```
procedure MANDIST(state)                      ▷ The current puzzle configuration
    total ← 0
    puzzleLength ← state.size()
    dimensions ← √puzzleLength
    for i ← 1, puzzleLength do              ▷ Loops through each tile of the puzzle
        tileValue ← state[i]
        expectedRow ← (tileValue − 1) ÷ dimensions
        expectedCol ← (tileValue − 1) mod dimensions
        rowNum ← i ÷ dimensions
        rowNum ← i mod dimensions
        total ← total+ | expectedRow − rowNum | + | expectedCol − colNum |
    return total                                      ▷ The heuristic is the total
```

Algorithm 5 Iterative Deepening A Star

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

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

procedure DFS(state, bound)
    if f(state) > bound then
        return f(state)
    if h(state) = 0 then                   ▷ No more moves needed to reach goal state
        return solved
    min ← ∞
    for neighbour in neighbours(state) do
        temp ← DFS(neighbour, bound)
        if temp < min then
            min ← temp
    return min                             ▷ Returns the smallest of the neighbours
```

Algorithm 6 Breadth-First Search

```
procedure BFS(state)
    s ← empty set
    q ← empty queue
    q.add(state)
    q.enqueue(state)
    while q.size() > 0 do
        currentState ← q.dequeue()
        if currentState = goal then
            return current
        for neighbour in neighbours(currentState) do
            if neighbour is not in s then
                s.add(neighbour)
                q.enqueue(neighbour)
```

Algorithm 7 Is Current State Solvable

```
procedure IsSolvable(state)
    puzzleLength ← state.size()
    gridWidth ← √puzzleLength
    blankRowEven ← true
    for i ← 1, puzzleLength do
        if state[i] = 0 then
            blankRowEven ← (i/gridWidth) mod 2 = 0
            continue
            for j ← i + 1, puzzleLength do
                if state[i] > state[j] and state[j] ≠ 0 then
                    parity ←!parity
    if gridWidth mod 2 = 0 and blankRowEven then
        return !parity
    return parity
```

Algorithm 8 Iterative Deepening Depth-First Search

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

procedure IDDFS(state)
    for depth ← 0, ∞ do
        found ← DLS(state, depth)
        if found ≠ null then
            return found
procedure DLS(state, depth)
    if depth = 0 and state = goal then
        return found
    if depth > 0 then
        for neighbour in neighbours(state) do
            found ← DLS(neighbour, depth − 1)
            if found ≠ null then
                returnfound
    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.