AI Machine Problem

Generated by Doxygen 1.9.5

Chapter 1

Machine Problem

1.1 Introduction: BSCS 3B AI PROJECT

This project is led by Kurt Mejorada, with Karl Francis Catolico running the machine and finding bugs while Paolo Cuenca was tasked with documenting and providing analysis to the given in trying to program both blind and heuristic search algorithms.

1.2 Instructions

Write a program in C/C++ implementing a blind search strategy, i.e. Iterative Deepening Search(IDS) and a heuristic search strategy, i.e. A$*$ Search with graph search to solve the 8- puzzle problem using Manhattan distance as the heuristic. Your program should use the board configuration below as the goal state and lets the user input the initial/start board configuration.

1	2	3
8		4
7	6	5

- It should output the following for both the IDS and A$*$ Search:
  1. solution path(corresponds to the moves needed to reach the goal): e.g. [Up-Left-Left-Right]
  2. solution cost(# of moves in the solution): 4
  3. number of nodes expanded before reaching the goal
  4. running time

1.3 Analysis

Initial State				IDS	A*
Easy			Solution path	U R U L D	U R U L D
1	3	4	Solution cost	5	5
8	6	2	Number of nodes expanded	117	5
7		5	Running Time	0.016	0
Medium			Solution path	U R R D L L U R D	U R R D L L U R D
2	8	1	Solution cost	9	9
	4	3	Number of nodes expanded	992	17
7	6	5	Running Time	0.01	0
Hard			Solution path	LLURDLURDLUURRDLLURD	L U L U R RDLURD
2	8	1	Solution cost	20	12
4	6	3	Number of nodes expanded	23848	26
7	5		Running Time	0.422	0
Worst			Solution path	LDRRUULLDDRRUULLDDRRUULLDDRRUL	ULDDRRUULLDDRRUULLDDRRUULLDDRU
5	6	7	Solution cost	30	30
4		8	Number of nodes expanded	213565	940
3	2	1	Running Time	45.789	0.01

Your preferred initial configuration			Solution path	U U L L D D R R U L L U R R D D L U	L L U U R D D R U U L L D D R U
1	2	4	Solution cost	18	16
8	3	5	Number of nodes expanded	16716	200
6	7		Running Time	0.144	0.002

During the multiple runs executed for both A* and IDS algorithm. It can be inferred that there was not much difference in configurations for the easy and medium categories but as we moved on to more difficult ones, the time it took for IDS significantly increased. This can be also be observed from the huge difference between the nodes expanded for the two algorithms, thus having longer running times.

There were also situations where both algorithms had the same solution cost but IDS would take much longer. From the multiple configurations tested, the results show that A* was better for all these configurations, still the use of IDS would not be much different for lower difficulty configurations of the 8-puzzle.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 list Struct Reference

Data structure for creating a linked-list of states.

Public Attributes

- puzzle * state
- list * next

4.1.1 Detailed Description

Data structure for creating a linked-list of states.

4.1.2 Member Data Documentation

4.1.2.1 next

```
list* list::next
```

4.1.2.2 state

```
puzzle* list::state
```

4.2 List Class Reference

This class manipulates the values of states.

Public Member Functions

- puzzle * front ()

  Accessing a node (state) in the BEGINNING of the list & popping it afterwards.
- puzzle * end ()

  Accessing a node (state) in the END of the list & popping it afterwards.
- void insertFront (puzzle *s)
- void insertEnd (puzzle *s)
- bool notListed (puzzle *state)

  Helps preventing insertion of the same node twice into the list.
- puzzle * pickBestState ()
- void view ()

Public Attributes

- list * lst = NULL

4.2.1 Detailed Description

This class manipulates the values of states.

4.2.2 Member Function Documentation

4.2.2.1 end()

```
puzzle * List::end ( )  [inline]
```

Accessing a node (state) in the END of the list & popping it afterwards.

4.2.2.2 front()

```
puzzle * List::front ( )  [inline]
```

Accessing a node (state) in the BEGINNING of the list & popping it afterwards.

4.2.2.3 insertEnd()

```
void List::insertEnd (
            puzzle * s )  [inline]
```

4.2.2.4 insertFront()

```
void List::insertFront (
            puzzle * s )  [inline]
```

4.2.2.5 notListed()

```
bool List::notListed (
            puzzle * state )  [inline]
```

Helps preventing insertion of the same node twice into the list.

Parameters

state	

Returns

FALSE or **TRUE** if the given state is already in the list or not.

4.2.2.6 pickBestState()

```
puzzle * List::pickBestState ( )  [inline]
```

Chooses the state on the entire list with the lowest heuristic value.

4.2.2.7 view()

```
void List::view ( )  [inline]
```

View the entire list of states.

4.2.3 Member Data Documentation

4.2.3.1 lst

```
list* List::lst = NULL
```

4.3 puzzle Struct Reference

The main data structure for storing a state.

Public Attributes

- int board [SIZE][SIZE]
- Vector blankTile
- int level

  Depth of a node used in IDS.
- char move

  Holds the previous tile movement.
- int manhattanDistance

  This is used in astar search.
- puzzle * parent

  Pointer to the parent node.

4.3.1 Detailed Description

The main data structure for storing a state.

This is in visualization of a puzzle.

4.3.2 Member Data Documentation

4.3.2.1 blankTile

```
Vector puzzle::blankTile
```

4.3.2.2 board

```
int puzzle::board[SIZE][SIZE]
```

4.3.2.3 level

```
int puzzle::level
```

Depth of a node used in IDS.

4.3.2.4 manhattanDistance

```
int puzzle::manhattanDistance
```

This is used in astar search.

4.3.2.5 move

```
char puzzle::move
```

Holds the previous tile movement.

4.3.2.6 parent

```
puzzle* puzzle::parent
```

Pointer to the parent node.

4.4 Vector Class Reference

Creating objects which keeps the position of the blank tile for each state.

Public Member Functions

- void setIndex (int x, int y)

Public Attributes

- int i
- int j

4.4.1 Detailed Description

Creating objects which keeps the position of the blank tile for each state.

4.4.2 Member Function Documentation

4.4.2.1 setIndex()

```
void Vector::setIndex (
            int x,
            int y )  [inline]
```

Parameters

i	is the x value
j	is the y value

4.4.3 Member Data Documentation

4.4.3.1 i

```
int Vector::i
```

4.4.3.2 j

```
int Vector::j
```

Chapter 5

File Documentation

5.1 MEJORADA_CATOLICO_CUENCA_PROJECTINAI_3B.cpp File Reference

```
#include <iostream>
#include <time.h>
```

Classes

- class Vector

  Creating objects which keeps the position of the blank tile for each state.
- struct puzzle

  The main data structure for storing a state.
- struct list

  Data structure for creating a linked-list of states.
- class List

  This class manipulates the values of states.

Macros

- #define SIZE 3
- #define MAXDEPTH 12

Functions

- puzzle * newState (int state[ ][SIZE])

  function declarations.
- puzzle * newInitialState (int arr[ ][SIZE])

  the state that accepts array which contains tile arrangement.
- puzzle * move (puzzle *state, char direction)

  movement of the blank tile
- bool movable (puzzle *state, char direction)

checks if it is a valid move for the blank tile

- bool isEqual (puzzle *state1, puzzle *state2)

  checks the visited list if the state has already been on it – notListed() function
- bool isGoal (puzzle *state1)

  checks if the goal has been found through comparison
- void printState (puzzle *state)
- int printStates (puzzle *state)
- int getManhattanDistance (puzzle *state)
- void Astar (puzzle *initialState)

  A Search Algorithm.*
- void IDS (puzzle *initialState)

  IDS Search Algorithm.
- int main ()

  MAIN FUNCTION.
- int abs (int x)
- int distBetween2Tiles (puzzle *state, Vector correctTile)

  Used in getManhattanDistance function.

Variables

- int goalState [ ][SIZE] = {{1, 2, 3}, {8, 0, 4}, {7, 6, 5}}

  Set for global variables.

5.1.1 Macro Definition Documentation

5.1.1.1 MAXDEPTH

```
#define MAXDEPTH 12
```

Note

This is done to prevent weaker machines from crashing as the IDS algorithm runs

5.1.1.2 SIZE

```
#define SIZE 3
```

5.1.2 Function Documentation

5.1.2.1 abs()

```
int abs (
            int x )
```

Returns

the absolute value of the integer | abs = absolute

5.1.2.2 Astar()

```
void Astar (
            puzzle * initialState )
```

A∗ Search Algorithm.

5.1.2.3 distBetween2Tiles()

```
int distBetween2Tiles (
            puzzle * state,
            Vector correctTile )
```

Used in getManhattanDistance function.

5.1.2.4 getManhattanDistance()

```
int getManhattanDistance (
            puzzle * state )
```

Calculates the distance between 2 tiles (goalstate and misplaced tile)

Note

useful for getting the state with the lowest heuristic value and records the value

5.1.2.5 IDS()

```
void IDS (
            puzzle * initialState )
```

IDS Search Algorithm.

Warning

This may cause crashing for weaker machines when executed.

5.1.2.6 isEqual()

```
bool isEqual (
            puzzle * state1,
            puzzle * state2 )
```

checks the visited list if the state has already been on it – notListed() function

5.1.2.7 isGoal()

```
bool isGoal (
            puzzle * state1 )
```

checks if the goal has been found through comparison

5.1.2.8 main()

```
int main ( )
```

MAIN FUNCTION.

Initialization of various difficulties to choose from

computation for running time

5.1.2.9 movable()

```
bool movable (
            puzzle * state,
            char direction )
```

checks if it is a valid move for the blank tile

5.1.2.10 move()

```
puzzle * move (
            puzzle * state,
            char direction )
```

movement of the blank tile

actions of the blank tile | U = Up; R = Right; D = Down; L = Left

5.1.2.11 newInitialState()

```
puzzle * newInitialState (
            int arr[][SIZE] )
```

the state that accepts array which contains tile arrangement.

initial state has a g(x) = 0. S = Starting move.

Returns

the created state. creation of new initial state

5.1.2.12 newState()

```
puzzle * newState (
            int state[][SIZE] )
```

function declarations.

function implementations

new state that accepts array that contains tile arrangement finds the blank tile

by default sets to -1 to determine if it is not yet calculated

5.1.2.13 printState()

```
void printState (
            puzzle * state )
```

5.1.2.14 printStates()

```
int printStates (
            puzzle * state )
```

5.1.3 Variable Documentation

5.1.3.1 goalState

```
int goalState[][SIZE] = {{1, 2, 3}, {8, 0, 4}, {7, 6, 5}}
```

Set for global variables.