Bicol University
College of Science
1st Semester, 2021-2022

C++ Implementation of IDS and A* search to solve the 8-Puzzle problem

Patrick Donghil (Code)
Raul Barquilla Jr. (Code and Documentation)
Ian Christian Lao (Documentation)
Roanne Pearl Misolas (Research)
BSCS – 3B

Arlene Satuito
Professor

I. Overview

A C++ Implementation of blind search strategy (IDS) and heuristic search strategy (A*) search to solve the 8-Puzzle problem. This program uses the board configuration below as the goal state.

1	2	3
8	0	4
7	6	5

For both the IDS and A* Search, the program outputs the following:

• Solution Path (corresponds to the moves needed to reach the goal): e.g. [Up-Left-Left-Right]
• Number of nodes expanded
• Solution Cost
• Running Time

Note: Since we used C++ language to implement the 8-puzzle problem, to compile the program we used “g++ 8-puzzle.cpp -o 8-puzzle.exe” and to run: “./8-puzzle.exe”.

II. Source Code Description:

```cpp
class Vector2
{
public:
    int i, j;
    void setIndex(int x, int y)
    {
        i = x;
        j = y;
    }
};
```

this class is used for creating objects that keeps the position of the blank tile for each state

```cpp
struct eightPuzzle
{
    int board[n][n];
    Vector2 blankTile;
    int level;
    string move;
    int manhattanDistance;
    eightPuzzle *parent;
};
```

the main data structure for storing a state of a puzzle like

`` struct list { eightPuzzle *state; list *next; }; ```	data structure needed for creating a linked-list of states
`` void insertToFront(eightPuzzle *s) void insertToEnd(eightPuzzle *s) ```	accessing a node (state) in the end of the list & popping it afterwards
`` bool notInList(eightPuzzle *state) { list *tmplist = lst; while (tmplist != NULL) ```	this returns false or true if the given state is already in the list or not, this helps preventing insertion of the same node twice into the list
`` eightPuzzle *chooseBestState() { list *tmplist = lst; list *previous; list *survivor; eightPuzzle *bestState = NULL; ```	chooses the state on the entire list with the lowest heuristic value and holds the state with the lowest heuristic value
`` start = clock(); AStar(init); end = clock(); cpuTimeUsed = ((double)(end - start)) / CLOCKS_PER_SEC; ```	for measuring the running time

<table>
<tr><td>

```
void AStar(eightPuzzle *initialState)
{
    List openList;
    List closedList;
    openList.insertToFront(initialState);
    int counter = 0;
```

</td><td>A* search function</td></tr>
<tr><td>

```
void IDS(eightPuzzle *initialState)
{
    int i = 0, counter = 0;
    while (true)
    {
        List closed;
        List stack;
        stack.insertToFront(initialState);
```

</td><td>IDS function</td></tr>
<tr><td>

```
eightPuzzle *move(eightPuzzle *state, string direction)
{
    eightPuzzle *tmp = newState(state->board);
    tmp->parent = state;
    tmp->level = state->level + 1;
```

</td><td>moves the blank tile in a certain direction, this determines the solution path from initial state to goal state</td></tr>
<tr><td>

```
int getManhattanDistance(eightPuzzle *state)
```

</td><td>for computing the number of moves to reach the goal</td></tr>
</table>

III. Analysis and Comparison of IDS and A* Search

Initial State		IDS	A*
Easy 1 3 4 8 6 2 7 _ 5	Solution Path	U R U L D	U R U L D
	Solution Cost	5	5
	Number of Nodes Expanded	117	5
	Running Time	0.008	0.008
Medium 2 8 1 _ 4 3 7 6 5	Solution Path	U R R D L L U R D	U R R D L L U R D
	Solution Cost	9	9
	Number of Nodes Expanded	992	17
	Running Time	0.016	0.008
Hard 2 8 1 4 6 3 7 5 _	Solution Path	L L U R D L U R D L U U R R D L L U R D	L U L U R R D L L U R D
	Solution Cost	20	12
	Number of Nodes Expanded	23848	26
	Running Time	0.875	0.016
Worst 5 6 7 4 _ 8 3 2 1	Solution Path	L D R R U U L L D D R R U U L L D D R R U U L L D D R R U L	U L D D R R U U L L D D R R U U L L D D R R U U L L D D R U
	Solution Cost	30	30
	Number of Nodes Expanded	213565	940
	Running Time	61.18	0.04
Random Input 1 3 2 4 0 8 7 6 5	Solution Path	L U R D R U L L D R R U R D	U R D L L U R R D L U L D R
	Solution Cost	14	14
	Number of Nodes Expanded	5686	96
	Running Time	0.08	0.008

IV. Conclusion

As we can see the comparison of the two algorithms from the given table, the time and space of A* search is more optimal than the time and space of the IDS. The execution time of A* search stay at millisecond from easy to worst case while IDS also took a millisecond in easy, medium, and hard case but it took 61.18 seconds in worst case and the IDS expanded nodes to reach the goal is much greater than the A* search.