School of Computing Sciences

Module: CMP-(6040A/7028A) Artificial Intelligence

Assignment: Solution of Tile problems

Set by : Pierre Chardaire e-mail: pc@uea.ac.uk
Date set : 5 October 2021
Value : 20%

Date due : 17 November by 15:00
Returned by : 16 December
Submission : on Blackboard

Learning outcomes

Apply techniques previously learned in the course to the solution of a search problem. Develop programming and problem-solving skills.

Specification

Overview

The aim of this coursework is to develop solutions for a classical AI search problem, to contrast the efficiency of the techniques implemented, and suggest how the techniques can be adapted to the solution of a variant of the problem.

Description

In this coursework you are asked to implement Depth First Search with Iterative Deepening (IDDFS) and Iterative Deepening A* (IDA*) for the solution of the n-tile problem. You are also asked to suggest how a variation of the problem could be solved efficiently.

1. Solution of n-Tile problem (programming in Python)

Using the state representation seen in class, a state corresponding to the configuration

1	2	3
4	5	6
7	8	

could be `[2,2,[[1,2,3],[4,5,6],[7,8,0]]]`.

You are asked to solve the following instances of the 8-tile problem in the smallest number of moves:

Instances with each of these start states:

```
[0, 0, [[0, 7, 1], [4, 3, 2], [8, 6, 5]]]
[0, 2, [[5, 6, 0], [1, 3, 8], [4, 7, 2]]],
[2, 0, [[3, 5, 6], [1, 2, 7], [0, 8, 4]]],
[1, 1, [[7, 3, 5], [4, 0, 2], [8, 1, 6]]],
[2, 0, [[6, 4, 8], [7, 1, 3], [0, 2, 5]]],
```

and goal state `[0, 2, [[3, 2, 0], [6, 1, 8], [4, 7, 5]]]`,

and instances with each of these start states:

```
[0, 0, [[0, 1, 8], [3, 6, 7], [5, 4, 2]]],
[2, 0, [[6, 4, 1], [7, 3, 2], [0, 5, 8]]],
[0, 0, [[0, 7, 1], [5, 4, 8], [6, 2, 3]]],
[0, 2, [[5, 4, 0], [2, 3, 1], [8, 7, 6]]],
[2, 1, [[8, 6, 7], [2, 5, 4], [3, 0, 1]]],
```

and goal state `[2,2,[[1,2,3],[4,5,6],[7,8,0]]]`.

Note that the last instance is the hardest instance in the sense that no other instance of the problem requires more moves.

Both codes, for IDDFS and for IDA*, should output the following for each instance:

- the number of moves to solve it (i.e. the number of states minus 1 in the shortest path from the instance to the goal),
- the number of `yield` of your `move` procedure made during the search for a solution, and
- the computing time the search took.

The code for IDDFS may reuse/adapt the `move` method seen in the lectures and adapt the DFS code provided. Note that you may have to deal with the fact that lists are passed by reference (i.e. deep copy may be needed).

The code for IDA* should rely on an h function made of the sum of evaluation functions of the numbered tiles (not including the 0 tile), where the evaluation of a tile is the Manhattan distance between its position in the current state and its position in the goal state. (This is the minimum number of moves a tile has to make to get into its final position.)

Your code should also work if the size of the problem is changed, i.e. a 15-tile problem is solved instead of an 8-tile problem.

2. Problem representation (problem-solving)

The objective of this part is not to provide code but explain how a variation of the n-tile problem (here called *new problem*) could be solved, assuming that you are addressing a programmer that has read your code for part 1 and is sufficiently intelligent and experienced.

First, in the new problem the rows in the grid wrap around, i.e. the last row of the grid is a neighbour of the first row. For example in the following configuration:

1	2	3
4	5	6
7	8	

the blank tile can be exchanged with tile number 3 to produce

1	2	
4	5	6
7	8	3

.

Question 1: What would be the changes to make to your IDDFS code of Part 1 to take into account this change to the n-tile problem specification?

In fact the new problem also allows an extra type of move that consists in shifting all the rows circularly down or up by one position. For example from the state

1	2	3
4	5	6
7	8	

a down shift would produce

7	8	
1	2	3
4	5	6

while an up shift would produce

4	5	6
7	8	
1	2	3

.

Also in this problem the cost of such shift moves is zero.

Question 2: How would you change your IDDFS code of part 1 to solve this problem *efficiently*?

Question 3: What physical design of an n-tile like puzzle would correspond to this problem?

Relationship to formative assessment

Extension of work on search methods seen in labs.

Deliverables

You are required to produce two independent Python programs to solve the 8-tile problem using

1. Iterative Deepening Depth First Search (program should be named `IDDFS.py`)
2. Iterative Deepening A* (program should be named `IDAstar.py`)

The codes must be clearly commented.

You must provide a comment at the start of each program that contains the results you were asked to output, that is the output of the solution method used in the program for each of the provided instances in the order they are listed in this document.

In addition, you have to answer questions 1, 2 and 3 of Part 2. The answers should be provided in a word document named `Part2.docx` that must be written using the template (not a word template) of the same name provided on Blackboard. The answer to this part should not exceed one page. The submission must stricly follow the specification of the provided template, i.e. fonts, font sizes, line spacing and header/footer/margin sizes should not be changed. Any change would result in a rejection of the document and a zero mark for Part 2.

Please, submit your two python sources files and your Word document as separate files rather than one zip file.

Resources

- Lecture notes.
- Labsheet solution on search.
- The Python tutorial: `https://docs.python.org/3.7/tutorial/index.html`

Marking scheme

The Iterative Deepening Depth First Search program is worth 20 marks with 10 marks for the correctness of the code, 5 marks for its efficiency and 5 marks for readibility (relevant comments are expected).

The Iterative Deepening A* program is worth 50 marks with 25 marks for correctness, 20 marks for efficiency and 5 marks for readibility.

The answers to Part 2 are worth 30 marks with 10 marks for Q1, 25 marks for Q2 and 5 marks for Q3.

However, although readibility of programs only counts for 10 marks, students may be further penalised if they do not provide sufficient comments, for example if they submit incomprehensible code, the correctness of which cannot be ascertained by the marker in reasonable time.