Assignment 1 - Search

Artificial Intelligence
WS 2015

Due: 2015-11-02, 23:55

General Information

Before you start, make sure you download the game framework version released for the first assignment from MOODLE. Empty classes and methods of the algorithms to be implemented are available in the package `at.jku.cp.ai.search.algorithms`.

You can test your implementations on a set of unit tests available in the package `at.jku.cp.ai.tests.assignment1`. The unit tests compare the results of your implementations to those of our reference implementations on the problem instances in `assets/assignment1`. Note that passing all these tests does not necessarily mean that your implementations have no flaws, since the tests do not cover all possible invariants!

For implementation details of the framework, please refer to the official *Rainbows And Unicorns Handbook* and/or source code comments where available.

Please answer any theoretical questions in a **PDF** file named "`assignment1/report.pdf`" and add this file to the **zip** file along with your code, as per the **formatting guidelines** of assignment 0.

Important:

- Make sure your code is **readable** and **understandable**.
- Add **comments** to explain what you are doing and why.
- When using non-primitive data types (List, Set, ...), **explain why you chose exactly this type over others!** (e.g. "We use LinkedList because we need fast insertion at the beginning and the end, which is O(1) for double linked lists.")

1 Theory

What is the runtime complexity of

- the “insert” operation at the end of a single linked list ?
- the “insert” operation at the end of a list backed by an array ?
- the “insert” operation of a hash table (hash set) ?
- the “contains” operation of a list ?
- the “contains” operation of a hash table (hash set) ?

Which data structure should we *actually* choose to implement a “closed list” as it is often called in the literature?

(1 point)

2 Uninformed Search

Implement the following uninformed search algorithms within the provided framework. Take care to avoid repeatedly expanding visited nodes as far as it is possible without increasing the big-O space complexity of the algorithms. If you need a priority queue, please use `at.jku.cp.ai.search.datastructures.StablePriorityQueue`, in order to make sure your expansion order is the same as ours. Also, process successor states returned by the `.adjacent()` method in the order they are returned (so, if you get `[x, y]` as successors, the algorithm should explore `x` first). Lastly, to keep track of costs for nodes, you will need additional data structures..

(A) Breadth-First Search
`src/at/jku/cp/ai/search/algorithms/BFS.java` (3 points)

(B) Uniform Cost Search
`src/at/jku/cp/ai/search/algorithms/UCS.java` (2 points)

(C) (Self-Avoiding) Depth-limited Depth-First Search
`src/at/jku/cp/ai/search/algorithms/DLDFS.java` (3 points)

(D) Iterative Deepening Search
`src/at/jku/cp/ai/search/algorithms/IDS.java` (1 points)

HINT: Read the handbook, section 4, “Search in graphs”

3 Heuristic Search

Implement the following heuristic search algorithms. If you need a priority queue, please use `at.jku.cp.ai.search.datastructures.StablePriorityQueue`, in order to make sure your expansion order is the same as ours.

(A) Greedy Best-First Search
`src/at/jku/cp/ai/search/algorithms/GBFS.java` (4 points)

(B) A* Search
`src/at/jku/cp/ai/search/algorithms/ASTAR.java` (4 points)

Consider the two heuristics implemented in the framework and used to test the algorithms. Answer the following questions, and **explain your answers** in detail:

(C) Which of the heuristics guarantees that Greedy Best-First Search will lead to an optimal solution? Which of them guarantees obtaining an optimal solution using A* Search? (1 points)

(D) Which of the heuristics is better? (1 points)

4 Create a Level

Look into the directory `assets/assignment1`. Each of its subdirectories contains a level that your implementations are tested against.

The `level` file contains a definition of the board, where '`#`' is a wall, '`.`' is a tile the player can move on, '`p`' is the player, and '`f`' is a fountain (the goal).

The `costs` file is of the same structure as the `level` file, but contains numbers for each reachable tile (everything except walls). These numbers correspond to the `cost` incurred by stepping on this tile.

The `*.path` files contain the valid paths returned by each of the search algorithms. For A* and GBFS, there are two files corresponding to the two different heuristics used. E.g. `astar_mh.path` contains the path that the A* implementation using the Manhattan Distance heuristic should return. Note that both the starting point and the goal position are included!

(A) Create a new level L101 (`level` file and `costs` file) for which each of the search algorithms {BFS, IDS, DLDFS, UCS} returns a **different path**. If this is fundamentally not possible for certain pairings of algorithms, explain **why**.
Create `*.path` files for every algorithm listed here. (2 points)

(B) Describe **how** all of the search algorithms in this list {BFS, IDS, DLDFS, UCS, GBFS, ASTAR} compute their solution. You can do this e.g. by drawing the **search space** as a tree, clearly **indicating the expansion order** of the nodes and including all necessary information like accumulated costs, etc... (6 points)

HINT: create a small board.

Include the `level` and the `costs` file in your submission, in a directory called "`assignment1/L101`". Any drawings and textual description should go into "`assignment1/report.pdf`".