Trabalho prático individual nº 1

Inteligência Artificial / Introdução à Inteligência Artificial
Ano Lectivo de 2023/2024

28-29 de Outubro de 2023

I Important remarks

1. This assignment should be submitted via *GitHub* within 27 hours after the publication of this description. The assignment can be submitted after 27 hours, but will be penalized at 5% for each additional hour.
2. Complete the requested functions in module `"tpi1.py"`, provided together with this description. Keep in mind that the language adopted in this course is Python3.
3. Include your name and number and comment or delete non-relevant code (e.g. test cases, print statements); submit only the mentioned module `"tpi1.py"`.
4. You can discuss this assignment with colleagues, but you cannot copy their programs neither in whole nor in part. Limit these discussions to the general understanding of the problem and avoid detailed discussions about implementation.
5. Include a comment with the names and numbers of the colleagues with whom you discussed this assigment. If you turn to other sources, identify those sources as well.
6. All submitted code must be original; although trusting that most students will do this, a plagiarism detection tool will be used. Students involved in plagiarism will have their submissions canceled.
7. The submitted programs will be evaluated taking into account: performance; style; and originality / evidence of independent work. Performance is mainly evaluated concerning correctness and completeness, although efficiency may also be taken into acount. Performance is evaluated through automatic testing. If necessary, the submitted modules will be analyzed by the teacher in order to appropriately credit the student's work.

II Exercices

Together with this description, you can find the module `tree_search`. You can also find attached the modules `cidades`, containing the `Cidades(SearchDomain)` class. These modules are similar to the ones initially provided for the practical classes, but with some changes and additions, namely:

- Terminal and non-terminal nodes are being counted.
- Loop prevention (repeated states along a path) is implemented
- The `Cidades(SearchDomain)` is given fully implemented.

Don't change the `tree_search` and `cidades` modules.

The module `tpi1_tests` contains several test cases. If needed, you can add other test code in this module.

Module `tpi1` contains the classes `MyNode(SearchNode)` and `MyTree(SearchTree)`. There is also a new domain `OrderDelivery(SearchDomain)`. In the following exercices, you are asked to complete certain methods in these classes. All code that you need to develop should be integrated in the module `tpi1`.

1. Create a new method `search2()` in class `MyTree`, similar to the original search method, and make sure that nodes (class `MyNode`) have the attributes `depth`, `cost` and `heuristic`, with the usual meaning, and also `eval`, initially given by the sum of cost and heuristic.

2. Implement the method astar_add_to_open(lnewnodes), which updates the queue of nodes according to the values of the `eval` attribute. In the case of a tie in `eval` values, use the alphabetical order of cities to break the tie.

3. Develop a method manage_memory() in class `MyTree()` to keep memory usage under a given threshold. Specifically, this method should be called immediately after the expansion of a new node in the following conditions: the selected strategy is `'A*'`; a value was given for parameter `maxsize` in the `MyTree()` constructor; and the current tree size (i.e. total number of nodes) exceeds `maxsize`. When the threshold is exceeded, some nodes are marked for deletion. When all children of a parent node are marked for deletion, they are finally removed from the queue and that parent node is returned to the queue. The procedure is the following: After the queue is sorted, search for nodes that are not yet marked for deletion, starting from those with highest `eval`, i.e. from the end of the queue. When one is found, mark this node for deletion and check if all its siblings are also marked for deletion. If that is the case, the node and siblings are removed and teir parent goes back to the queue. At this point, the `eval` of the parent should be updated with the minimum `eval` value that existed in the removed child nodes. If the tree size still exceeds the threshold, this procedure is repeated one or more times as needed. Call manage_memory() in a suitable location in `search2()`.

4. The search domain `OrderDelivery` remains mostly to be implemented, except for the `actions()` method which is already given. The purpose of this domain is to support path planning for order delivery, e.g. in a company that delivers orders to shops, or in a shop that delivers orders to end customers. In this type of domains, the start and end locations are the same, and other cities may also be repeated along the path. You have to choose the exact representations for states, but note that each state should be a tuple and the first element of that tuple should be the current city. Implement all the required methods in this domain. You are free to implement any heuristic. A part of your score will be given according to the number of nodes that the search generates (as you know, better heuristics lead to less nodes).

5. Implement the wrapper function `orderdelivery_search (domain,city , targets , strat ,maxsize)`. This functions creates the problem and search tree, calls the `search2()` method, and finally returns the required result.

III Clarification of doubts

This work will be followed through `http://detiuaveiro.slack.com`. The clarification for the main doubts will be added here.

1. .....

   Resposta: .....