

Trabalho prático individual nº 1

Inteligência Artificial / Introdução à Inteligência Artificial Ano Lectivo de 2020/2021

13-14 de Novembro de 2020

I Observações importantes

1. This assignment should be submitted via *GitHub* within 32 hours after the publication of this description. The assignment can be submitted after 32 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 assignment. 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 account. 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`, similar to the one initially provided for the practical classes, but with small changes and additions, namely:

- A new abstract method `middle` in the class `SearchDomain`.
- A new attribute `children` (list of child nodes) in the class `SearchNode`.
- New attributes `terminal`, `non_terminal` and `root` in the class `SearchTree`.
- New method `show()`, in the class `SearchTree` which displays the full structure and content of a generated search tree.

You can also find in this assignment the modules `cidades` and `strips`, containing similar classes to those used in the practical classes. Don't change the `tree_search`, `cidades` and `strips` 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 `MyTree(SearchTree)`, `MySTRIPS(STRIPS)`, `MinhasCidades(Cidades)`. In the following exercises, 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. In class `MySTRIPS`:
 - (a) Implement the method `result(state,action)`, which is supposed to return a list of predicates describing the new state produced by `action` in `state`.
 - (b) Suppose that Python didn't have the data type `set`, that we can use in the `strips` module to represent states. Suppose also that we use lists to represent STRIPS planning states, as in the previous exercise. In this case, how could we easily find out if two states are equivalent? One possibility is to sort the lists representing the two states and then see if they are equal. This is done by method `equivalent(state1,state2)` in class `STRIPS`. However, you need to implement the method `sort(state)` in class `MySTRIPS`.
2. In class `MinhasCidades`, implement the method `middle(state1,state2)` which determines the state `m` which minimizes the sum of the heuristic values `heuristic(state1,m)` and `heuristic(m,state2)`.
3. Two uninformed search strategies are well known: *breadth-first search* and *depth-first search*. They have well known advantages and disadvantages. In the following exercises you will implement two new strategies that you can see as fusions of those basic strategies in an attempt to retain the advantages of both and avoid as much as possible their disadvantages. All required methods are to be implemented in the class `MyTree`.
 - (a) Implement the method `hybrid1.add_to_open(lnewnodes)` which adds those nodes in even position of `lnewnodes` to the front of the `open_nodes` queue and the remaining nodes to the end of the queue.
 - (b) Implement the method `hybrid2.add_to_open(lnewnodes)` which adds the nodes in `lnewnodes` to the `open_nodes` queue in such a way that the nodes in the queue are always sorted by `depth-offset`. In this context, the `offset` of a node is its position among the nodes at the same `depth`, considering that the leftmost node (i.e. the node first created at that depth) is at position 0 (zero).
 - (c) Implement method `search2()` by copying the standard `search()` method from the `tree_search` module and adding code to include in each node its `depth` and `offset`.

4. When search spaces are large, one possibility is to use some sort of divide and conquer strategy. Implement in the class `MyTree` the method `search_from_middle()` which determines the middle state (by calling a previously implemented method, see above) and diving the given search problem in two search problems (one from the initial to the middle state, and the other from the middle state to the goal state), solving them separately and concatenating the respective solutions. The method should store the two auxiliary search trees in attributes `from_init` and `to_goal` of the main search tree where the method is called.

III Clarification of doubts

This work will be followed through <http://detiuaveiro.slack.com>. The clarification of the main doubts will be placed here.

1. O professor tem as cotações das perguntas??

Resposta: Já sabem que, globalmente é assim: correcção e completude (desempenho) - 70%; estilo do código - 10%; originalidade - 20%. Dentro dos 70% do desempenho: 1=10%, 2=15%, 3=45%, 4=30%.

2. No exercício 2 não é suposto retornar uma cidade em vez de state?

Resposta: No domínio das cidades, o estado é uma cidade, por isso está tudo bem.

3. No exercício 1, é normal as 4 linhas de output poderem surgir por uma ordem diferente certo?

Resposta: O módulo STRIPS usa dicionários, por isso não há uma ordem fixa.

4. Podemos importar packages built-in?

Resposta: Sim.

5. No exercício 3.c) pede-se para "include in each node its depth and offset"; para isto não teríamos de alterar a classe `SearchNode` para aceitar o `depth` e o `offset` como argumentos no construtor?

Resposta: Alterar o `SearchNode`, não podem! No entanto, há alternativas.

6. No 3.c) devemos criar uma subclasse de `SearchNode` que tenha os atributos `depth` e `offset`?

Resposta: Fica ao vosso critério; há várias alternativas.

7. No 1.b), qual é o critério de ordenação das listas?

Resposta: O critério que está no ficheiro de resultados é alfabético, mas podem usar outro critério qualquer, desde que não gere ambiguidade

8. Podia explicar melhor o depth offset?

Resposta: Não é depth offset; são duas coisas diferentes; `depth` é a profundidade, com usámos nas aulas práticas; `offset` é posição de cada nó no respectivo nível. Por exemplo, o offset da raiz é 0; se a raiz tiver 3 filhos, os respectivos offsets vão ser 0, 1, 2; se cada filho tiver dois filhos, o terceiro nível vai ter 6 nós, com `offset` de 0 a 5, numerados pela ordem em que são criados.

9. No 3.a), os elementos de `lnewnodes` que estão em índices pares passam para a frente da fila e os restantes vão para o final certo?

Resposta: Correcto!

10. No 3.b), a ordenação deve ser por ordem crescente de profundidade e no caso desta ser igual, de `offset`?

Resposta: Não, é para ordenar pelo valor resultante da subtracção, como está no enunciado. Notem que a 3.b) depende da 3.c), porque vai ser a 3.c) que vai registar a informação necessária nos nós; na 3.b) vai garantir que os nós estão sempre ordenados por $D-O$, em que D é a `depth` e O é o `offset`.

11. Podemos usar funções do tipo `sorted()` de Python no exercício 2 de Strips?

Resposta: Podem.

12. O enunciado diz que o `middle()` entre Braga e Faro é Águeda, mas não existe nenhuma ligação directa de Braga para Águeda.

Resposta: Não tem que haver ligação; a escolha é feita com base na heurística.

13. Podemos alterar o `__init__()` da `MyTree`?

Resposta: sim

14. Listas de compreensão contam pontos em estilo de código? Ou legibilidade e limpeza do mesmo apenas?

Resposta: Os aspectos de elegancia e clareza são importantes; verei também as funções de ordem superior do Python.

15. Na 1.b) é necessário que, quando por exemplo há vários `Floor`, ver também a letra que está dentro deles para a ordenação?

Resposta: É necessário haver uma ordenação não ambígua, seja ela qual for. Tem que servir para ver se dois estados são iguais comparando directamente: `sort(s1)==sort(s2)` como está no `equivalent()`.

16. No exercício 4, no fim de criarmos a lista, devemos limpar estados desnecessários?

Resposta: Não, retornam o caminho que sair da pesquisa (ou das pesquisas); é melhor ou pior conforme a estratégia. Notem que, por defeito, a pesquisa é em largura; é o que está ser usado nos testes do ex. 4; mas podia usar-se outra estratégia. É para dividir em dois subproblemas, mas não é para fazer isso recursivamente!

17. No exercício 3.a), a ordem pela qual os elementos pares são introduzidos no início da `open_nodes` é importante certo? se é do ultimo para o primeiro ou vice versa?

Resposta: O enunciado é pouco explícito nesse ponto, por isso o que manda é o que está no `results.txt`; faça de maneira a dar igual.

18. No ex. 4, o `solution` neste momento fica com `None`. Temos que atribuir um valor a este atributo?

Resposta: Sim, existindo um nó solução, deve lá ser colocado. O `None` significa inexistência. Têm que ver quando é que se aplica e quando não se aplica.