LINGI2261: Artificial Intelligence
Assignment 1: Solving Problems with Uninformed Search

Gael Aglin, Alexander Gerniers, Yves Deville

September 23, 2019

Guidelines

- This assignment is due on **Wednesday 09 October, 18:00**.
- *No delay* will be tolerated.
- *Document* your source code (at least the difficult or more technical parts of your programs). Python docstrings for important classes, methods and functions are also welcome.
- Indicate clearly in your report if you have *bugs* or problems in your program. The online submission system will discover them anyway.
- Copying code or answers from other groups (or from the internet) is strictly forbidden. Each source of inspiration must be clearly indicated.
- Source code shall be submitted on the online *INGInious* system. Only programs submitted via this procedure will be graded. No report or program sent by email will be accepted.
- Respect carefully the *specifications* given for your program (arguments, input/output format, etc.) as the program testing system is *fully automated*.

Deliverables

- The following files are to be submitted on *INGInious* inside the *Assignment 1* task(s):
  - `knight.py`: The file containing your implementation of the Knight's tour solver. Your program should take four arguments. The first two are respectively the number of columns and the number of rows of the chessboard while the last two represent respectively y and x coordinates of the starting position of your knight. It should print a solution to the problem to the standard output, respecting the format described further. The file must be encoded in **utf-8**.
  - `report_A1_group_XX.pdf`: Answers to all the questions in a single report, named. Remember, the more concise the answers, the better.

Anti plagiat charter

As announced in the class, you'll have to electronically sign an anti plagiat charter. This should be done ***individually*** in the **INGInious** task entitled *Assignment 1: Anti plagiat charter.* Both students of a team must sign the charter.

Important

- For the implementation parts of the assignments, you are required to use *Python 3*
- Python 3.7.4 can be downloaded at `https://python.org/downloads/`
- On your computer, after installing Python 3, you will be able to launch your programs using *python3 <your_program>*
- In the labs, you can find Python 3 under */opt/python3/bin/python3*. You can of course add */opt/python3/bin* to your PATH to be able to launch your programs using *python3 <your_program>*.
- Python 3 documentation can be found at `http://docs.python.org/py3k/`
- The assignments must be submitted via the *INGInious* tool. You first have to *create groups of two*. Only then you will be granted access to the submission tool. The procedure to register and submit your hard work is described below. *Don't wait until the last minute* to create your group and to familiarize with the tool.
- If you want to ask us questions, we are on Tuesdays from 2:30pm to 4:30pm, in office A 158 (Gaël) at Réaumur. *For general questions, do not hesitate to use the forum set up espacially for this assignment, on Moodle*.

Submitting your programs

Python programs must be submitted on the INGInious website: `https://inginious.info.ucl.ac.be`. In order to do so, you must first create groups of two. To do so, assign yourself in an available group on the INGInious page of the course. Inside INGInious, you can find different courses. Inside the course 'LINGI2261: Artificial Intelligence', you will find the tasks corresponding to the different assignments due for this course. The task at hand for this assignment is *Assignment 1: Knight's tour*. In the task, you can submit your program (one python file containing the knight's tour solver, encoded in utf-8). Once submitted, your program will immediately be evaluated on the set of given instances and also on a hidden set of instances. The results of the evaluation will be available directly on INGInious. You can, of course, make as many submissions as you want. For the grade, only the last fully correct submission (or the last submission if no fully correct submission has been made) will be used. You thus know the grade you will receive for the program part of the assignment! If you have troubles with INGInious, use the dedicated forum on Moodle.

Important

Although your programs are graded automatically, they will still be checked for plagiarism !

1 Python AIMA (3 pts)

Many algorithms of the textbook "AI: A Modern Approach" are implemented in Python. Since you are required to use Python 3, we will provide a Python 3 compliant version of the AIMA library. The Python modules can be downloaded on Moodle in *Documents (S2)*. All you have to do is to decompress the archive of aima-python3 and then put this directory in your python path: **export PYTHONPATH=path-to-aima-python3**. As we will use our own version of the library to test your programs, no modification inside the package is allowed.

The objective of these questions is to read, understand and be able to use the Python implementation of the *uninformed methods* (inside *search.py* in aima-python3 directory).

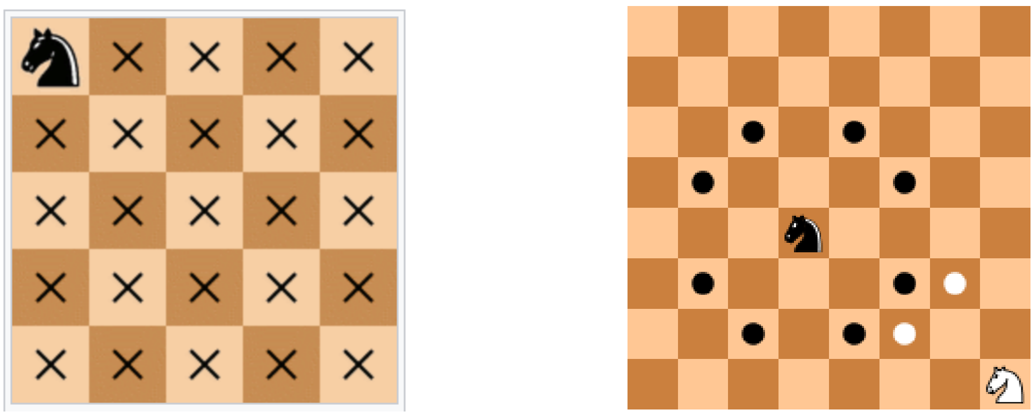
Questions

1. In order to perform a search, what are the classes that you must define or extend? Explain precisely why and where they are used inside a *tree_search*. Be concise! (e.g. do not discuss unchanged classes).
2. Both *breadth_first_graph_search* and *depth_first_graph_search* are making a call to the same function. How is their fundamental difference implemented (be explicit)?
3. In *graph_search* and *tree_search*, what is the effect of the instruction *fringe.extend(node.expand(problem))*. What are the classes and methods involved ?
4. What is the difference between the implementation of the *graph_search* and the *tree_search* methods and how does it impact the search methods?
5. What kind of structure is used to implement the *closed list*? What are the methods involved in the search of an element inside it? What properties must thus have the elements that you can put inside the closed list?
6. How technically can you use the implementation of the closed list to deal with symmetrical states? (hint: if two symmetrical states are considered by the algorithm to be the same, they will not be visited twice)

2 The Knight's tour Problem (17 pts)

The initial configuration is represented by a grid of shape $ncols \times nrows$ and a knight at its initial position index $[j][i]$ in the grid. j represents the rows axis while i represents the columns axis. The $[0][0]$ position is on the top left corner. You have to move the knight following the moves of a knight is a chess game such that all the cases of the chessboard must be visited only once. Figure 1b shows for the black knight all possible moves. Depending of the position of the white knight only two moves are possible. You can give more information on a knight moves on `https://en.wikipedia.org/wiki/Knight_(chess)`. In summary, a “knight's tour” is a sequence of moves of a knight on a chessboard such that the knight visits every square only once[1].

[1] `https://en.wikipedia.org/wiki/Knight%27s_tour`



(a) Screenshot of a complete 5×5 Knight's tour game interface.

(b) Screenshot of a knight's moves on a chessboard.

Figure 1: Illustrations of Knight's tour game

We provide on Moodle a set of 10 instances to test your implementation. These instances are part of instances on which your implementation will be evaluated on INGInious. Five (05) more hidden instances will be used.

One file is provided containing all the instances. Each line in the file represents an instance and will be passed individually to your implementation. The line representing an instance contains four (04) numbers described chronologically as following:

1. the number of columns of the board
2. the number of rows of the board
3. the position of the knight on rows axis
4. the position of the knight on columns axis

The solving of each instance involves the passing of the board through several states. We use utf-8 symbols in order to represent the tiles we see in Figure 1a.

- visited tile : ♞ *python: u"\u265E" — utf-8: 0xE2 0x99 0x9E*
- not yet visited tile : space character
- current position of the knight : ♘ *python: u"\u2658" — utf-8: 0xE2 0x99 0x98*

Each tile is separated by a space character and the borders of the board are represented by # characters. The following configuration represents the grid of shape 5×5 with five moves already done by the knight and the current position is at index [3][1]

```
###########
#      ♞  #
#  ♞      #
#        ♞#
#  ♘      #
#      ♞  #
###########
```

Figure 2: A state of Knight's tour problem

The output of the program should be a sequence of every intermediate grid, represented in the same way, starting with the initial state and finishing with the goal state. Figure 3 shows the solving process of instance: 5 5 1 1.

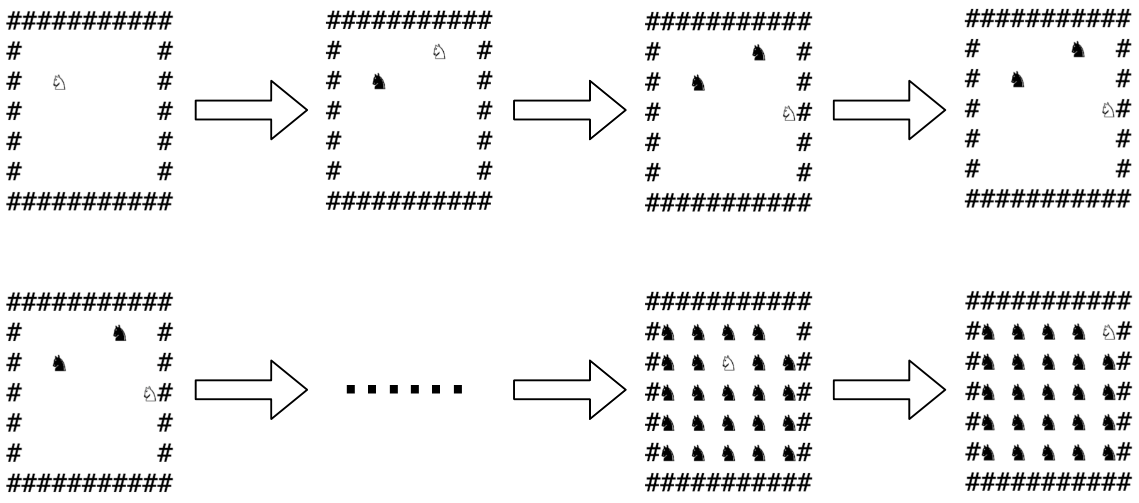


Figure 3: Solving process of instance : 5 5 1 1

You will implement at least one class *Knight(Problem)* that extends the class *Problem* such that you will be able to use search algorithms of AIMA. A small template (*knight.py*) is provided in the resources for this problem on moodle. Before diving into the code, we recommend you to first have a look at the questions below that need to be answered in your written report.

Questions

1. **Describe** the set of possible actions your agent will consider at each state. Evaluate the branching factor.
2. **Problem analysis.**
   (a) Explain the advantages and weaknesses of the following search strategies **on this problem** (not in general): depth first, breadth first
   (b) What are the advantages and disadvantages of using the tree and graph search **for this problem**. Which approach would you choose ?
3. **Implement** a Knight's tour solver in Python 3. You shall extend the *Problem* class and implement the necessary methods -and other class(es) if necessary- allowing you to test the following four different approaches:
   - *depth-first tree-search (DFSt)*;
   - *breadth-first tree-search (BFSt)*;
   - *depth-first graph-search (DFSg)*;
   - *breadth-first graph-search (BFSg).*

   Your file must be named `knight.py`.
4. **Experiments** must be realized (*not yet on INGInious!* use your own computer

or one from the computer rooms) with the provided 10 instances. Report in a table the results on the 10 instances for depth-first and breadth-first strategies on both tree and graph search (4 settings above). Run each experiment for a maximum of 3 minutes. You must report the time and the number of explored nodes to get a solution.

5. **Submit** your program (the `knight.py` file, encoded in **utf-8**) on INGInious. According to your experimentations, it must use the algorithm that leads to the best results. Your program must take as inputs the four number previously described separated by space character, and print to the standard output a solution to the problem satisfying the format described in Figure 3.
   Under INGInious (only 30s timeout per instance!), we expect you to solve at least 12 out of the 15 ones.

6. **Conclusion and further work.**

   (a) Are your experimental results consistent with the conclusions you drew based on your problem analysis (Q2)?

   (b) Which algorithm seems to be the more promising? Do you see any improvement directions for this algorithm? (Note that since we're still in uninformed search, *we're not talking about informed heuristics*).