LINFO1361: Artificial Intelligence
Assignment 1: Solving Problems with Uninformed Search

Charles Lohest, Eric Piette
February 2024

Guidelines

- This assignment is due on **Thursday 29 February, 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.
- 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 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*.
- The answer to questions must be given by filling in the latex template provided. The final file must be submitted on *gradescope*. No report sent by email will be accepted.
- Nothing must be modified in the template except your answer that you insert in the *answer* environments as well as your names and your group number. The names are provided through the command *students* while the command *group* is used for the group number. The dimensions of *answer* fields *must not* be modified either. For the tables, put your answer between the "&" symbols. Any other changes to the file will *invalidate* your submission.
- To submit on gradescope, go to `https://gradescope.com` and click on the "log in" button. Then choose the "school credentials" option and search for *UCLouvain Username*. Log in with your global username and password. Find the course LINFO1361 and the Assignment 1, then submit your report. Only one member should submit the report and add the second as group member.
- For those who have not been automatically added to the course, at the right bottom of gradescope homepage, ckick on "Enroll on course" button and type the code *86KJVB*.
- Check this link if you have any trouble with group submission `https://help.gradescope.com/article/m5qz2xsnjy-student-add-group-members`

Deliverables

- The following files are to be submitted:
  - `pacman.py` *(INGInious)*: The file containing your implementation of the PacMan solver. Your program should take one argument, the filepath containing the instance to solve. 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` *(Gradescope)*: Answers to all the questions using the provided template. 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 part of the assignment, you are required to use *Python 3*
- Python 3.9.1 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 `https://docs.python.org/3/`
- The coding assignment 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 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 available on Teams (Charles Lohest) on Tuesdays from 2:30pm to 4:30pm. *For general questions, do not hesitate to use the forum set up especially 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 'LINFO1361: 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: PacMan*. In the task, you can submit your program (one python file containing the Pacman 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 best submission will be considered. You thus know the grade you will receive for the program part of the assignment! If you have troubles with INGInious, use the assignment 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 the folder *Assignment 1* of *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. What is the difference between the implementation of the *graph_search* and the *tree_search* methods and how does it impact the search methods?
4. What kind of structure is used to implement the *reached nodes minus the frontier list*? What properties must thus have the elements that you can put inside the reached nodes minus the frontier list?
5. How technically can you use the implementation of the reached nodes minus the frontier 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 PacMan Problem (17 pts)

The Pacman problem is inspired by the famous video game Pacman which can be found here: `https://pacman.live/`. In this assignment, we will use a simplified version of the game without any enemies. The goal of this version will be to move on the grid and collect all the

fruits to complete the level. To collect a fruit you need to move to its location and stop on it. Once you collected the fruit the place becomes an empty box (".") The initial configuration is represented by a grid of shape $nrows \times ncols$ and a Pacman at its initial position index $[i][j]$ in the grid. i represents the rows axis while j represents the columns axis. The [0][0] position is in the top left corner. The # represents the wall that Pacman can not pass and the F represents the fruits that Pacman has to collect to win the game. Pacman can move following the horizontal direction or the vertical direction as much as he wants while there is no wall to stop him. The fruits need to be eaten by stopping on them in a minimal number of moves.

Each problem is described in an instance file. It contains the following information :

- The dimension of the grid
- The initial state where # are walls, P is Pacman and F are fruits

The goal state to reach will be when PacMan has eaten all the fruit on the grid.

The initial state is represented using ASCII symbols. There is no distinction between two tiles with the same ASCII character. The output of the program should be a minimal sequence of every intermediate grids, represented in the same way, starting with the initial state and finishing with the final state with no more fruit on the grid. Figure 2 shows a solution of the instance from Figure 1 in five moves. Of course, there may exist multiple solutions with the same minimal number of moves. We provide on Moodle a set of 10 instances to test your

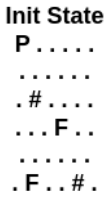

Figure 1: Example of an Init state where the grid is made of "." , Pacman is respresented by the "P" , the walls are represented by "" and the fruits are represented by the "F".

implementation. These instances are part of instances on which your implementation will be evaluated on INGInious. Five more hidden instances will be used. Format of the output The template file (Pacman.py) already implements the correct output format. We expect you to keep it unchanged. Each state is represented as depicted in Figure 2, i.e., the action to reach the current state (Init for the initial state), and the grid representation on the next line. There is a line-break between each state.

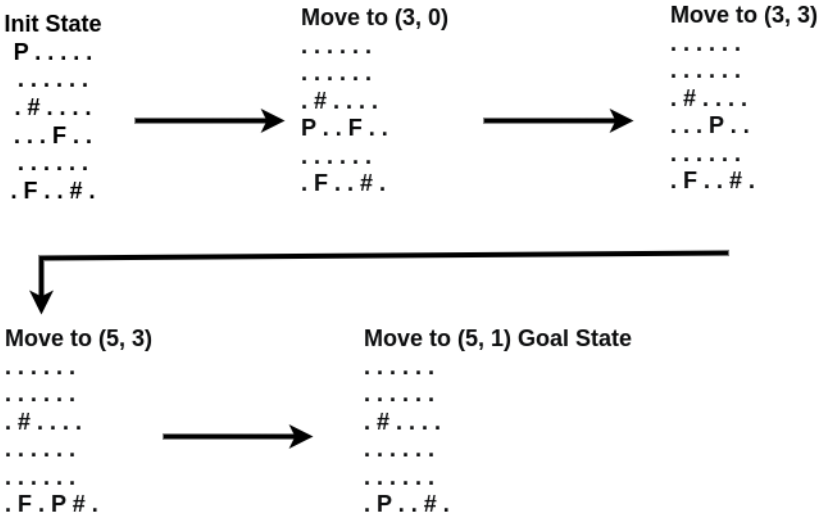

Figure 2: Representation of an optimal resolution of the problem in 4 moves

Questions

1. (a) **Describe** the set of possible actions your Pacman will consider at each state. Evaluate the branching factor.

   (b) How would you build the action to avoid the walls?

2. **Problem analysis.**

   (a) Explain the advantages and weaknesses of the following search strategies **on this problem** (not in general): depth first, breadth first. Which approach would you choose?

   (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 PacMan 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)*.

   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 1 minute. You must report the time, the number of explored nodes as well as the number of remaining nodes in the queue to get a solution.

4. **Submit** your program (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 only input the path to the instance file of the problem to solve, and print to the standard output a solution to the problem satisfying the format described earlier. Under INGInious (only 1 minute timeout per instance!), we expect you to solve at least 12 out of the 15 ones. Solving at least 12 of them will give you all the points for the implementation part of the evaluation.
5. **Conclusion.**
   (a) How would you handle the case of some fruit that is poisonous and makes you lose?
   (b) Do you see any improvement directions for the best algorithm you chose? (Note that since we're still in uninformed search, *we're not talking about informed heuristics*).