

LINGI2261: Artificial Intelligence

Assignment 2: Solving Problems with Informed Search

Lucía Mattenet, Gaël Aglin, Alexander Gerniers, Yves Deville
March 2022



Guidelines

- This assignment is due on **Thursday 10th March 2022, 18h00**.
- **No delay** will be tolerated.
- Not making a **running implementation** in **Python 3** able to solve (some instances of) the problem is equivalent to fail. Writing some lines of code is easy but writing a correct program is much more difficult.
- **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. The consequences of **plagiarism** is **0/20 for all assignments**.
- 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 2** task(s):
 - report_A2_group_XX.pdf: Answers to all the questions in a single report, named. Remember, the more concise the answers, the better.
 - The file `pagecollect.py` containing your Python 3 implementation of the Pacmen problem solver. Your program should take the path to the instance files as only argument. The search strategy that should be enabled by default in your programs is **A* with your best heuristic**. Your program should print the solution to the standard output in the format described further. The file must be encoded in **utf-8**.





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 2: Anti plagiat charter**. Both students of a team must sign the charter.

1 Search Algorithms and their relations (3 pts)

1.1 A^* versus uniform-cost search

Consider the maze problem given on Figure 1. The goal is to find a path from  to  moving up, down, left or right. The black positions represent walls. This question must be answered by hand and doesn't require any programming.



Questions

1. Give a consistent heuristic for this problem. Prove that it is consistent. Also prove that it is admissible. (1 pt)
2. Show on the left maze the states (board positions) that are visited during an execution of a uniform-cost graph search. We assume that when different states in the fringe have the smallest value, the algorithm chooses the state with the smallest coordinate (i, j) ($(0, 0)$ being the bottom left position, i being the horizontal index and j the vertical one) using a lexicographical order. (1 pt)
3. Show on the right maze the board positions visited by A^* graph search with a manhattan distance heuristic (ignoring walls). A state is visited when it is selected in the fringe and expanded. When several states have the smallest path cost, this uniform-cost search visits them in the same lexicographical order as the one used for uniform-cost graph search. (1 pt)

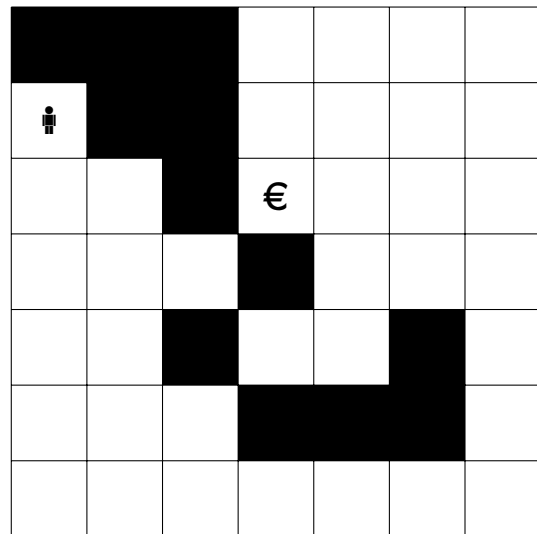
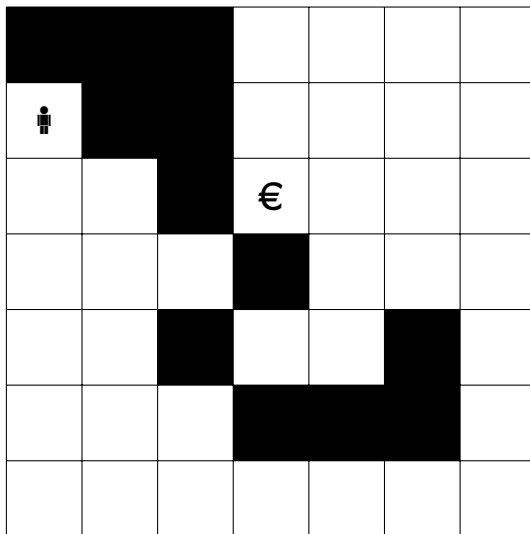


Figure 1

2 PageCollect problem (17 pts)

The problem you will solve for this assignment is the PageCollect problem. Again, the search procedures from `aima-python3` will help you solve it! In this problem, we model an unlucky student who finished printing their final report for a very hard class. On the way to the examiner, the student slips, and all of the pages of their report get scattered by the wind. The goal in this problem is to collect all the pages of the report around the city, and bring them to the examiner as fast as possible. The student can move north, west, south and east to collect the pages that are scattered in the world. The position of the examiner is marked in the world. The problem has an additional requirement: as the student cannot be late to the exam, the number of moves that they make must be minimal.

Input and output format

Your program must take one argument : the path to the file containing the initial state (e.g. `i01`). We use ASCII symbols in order to represent a state. Figure 2 shows an example of initial state, for the test instance `i01`.

```
#####
#      @      #
# p #        # # #
#   # # # # #
#   # # # # #
#   # # p # #
#   #   X # #
#   #   # # #
#####
```

Figure 2: The initial instance file `i01`.

The different symbols in this problem as follows:

- `@` marks the position of the student. There is only one student in this problem, so only one `@` will appear per instance of the problem.
- `p` indicates the position of a page of the student's report. There can be multiple pages, and the student has to collect all of them. Once the student moves over a page, the page is stored in the student's inventory and removed from the map of the world. Thus, in the state for a solution to this problem, no `p` remain on the world map.
- `X` marks the position of the examiner. There is only one examiner per instance of the problem. Once the student has collected all the pages `p`, she must go to the examiner.
- `#` marks a wall. The student cannot move to a position occupied by a wall.
- a space represents an empty position, on which the student can freely walk.

A solution to this problem is composed of the successive states of the game, induced by the displacement of the student, leading to the collection of all the pages and ending in the position of the examiner in a minimal number of steps.

The output of the program should be a minimal sequence of every intermediate grid, represented in the same way, starting with the initial state and finishing with the goal state, separated by an empty line ("`\n\n`"). Figure 3 gives an example of a valid solution for instance `i01`. This solution, obtained by executing `python3 pagecollect.py instances/i01`, is also optimal because it involves a minimal number of actions (or moves).

initial state

```
#####
#      @      #
# p #      # #
#   #      # #
#   #      # #
#   #      p #
#   #      X #
#####
```

state 1: west

```
#####
#      @      #
# p #      # #
#   #      # #
#   #      # #
#   #      p #
#   #      X #
#####
```

state 2: west

```
#####
#      @      #
# p #      # #
#   #      # #
#   #      # #
#   #      p #
#   #      X #
#####
```

state 3: west

```
#####
#      @      #
# p #      # #
#   #      # #
#   #      # #
#   #      p #
#   #      X #
#####
```

state 4: west

```
#####
# @      # #
# p #      # #
#   #      # #
#   #      # #
#   #      p #
#   #      X #
#####
```

state 5: south

```
#####
# @      # #
# p #      # #
#   #      # #
#   #      # #
#   #      p #
#   #      X #
#####
```

state 6: north

```
#####
# @      # #
# p #      # #
#   #      # #
#   #      # #
#   #      p #
#   #      X #
#####
```

state 7: east

```
#####
# @      # #
# p #      # #
#   #      # #
#   #      # #
#   #      p #
#   #      X #
#####
```

state 8: east

```
#####
# @      # #
#   #      # #
#   #      # #
#   #      # #
#   #      p #
#   #      X #
#####
```

state 9: east

```
#####
# @      # #
#   #      # #
#   #      # #
#   #      # #
#   #      p #
#   #      X #
#####
```

state 10: north

```
#####
# @      # #
#   #      # #
#   #      # #
#   #      # #
#   #      p #
#   #      X #
#####
```

state 11: north

```
#####
# @      # #
#   #      # #
#   #      # #
#   #      # #
#   #      p #
#   #      X #
#####
```

state 12: north

```
#####
#   #      # #
#   #      # #
#   #      # #
#   #      # #
#   #      p #
#   #      X #
#####
```

state 13: north

```
#####
#   #      # #
#   #      # #
#   #      # #
#   #      # #
#   #      p #
#   #      X #
#####
```

state 14: east

```
#####
#   #      # #
#   #      # #
#   #      # #
#   #      # #
#   #      p #
#   #      X #
#####
```

state 15: east

```
#####
#   #      # #
#   #      # #
#   #      # #
#   #      # #
#   #      p #
#   #      X #
#####
```

state 16: east

```
#####
#   #      # #
#   #      # #
#   #      # #
#   #      # #
#   #      @ #
#   #      X #
#####
```

state 17: east

```
#####
#   #      # #
#   #      # #
#   #      # #
#   #      # #
#   #      @ #
#   #      X #
#####
```

state 18: south

```
#####
#   #      # #
#   #      # #
#   #      # #
#   #      # #
#   #      @ #
#   #      X #
#####
```

goal state

```
#####
#   #      # #
#   #      # #
#   #      # #
#   #      # #
#   #      @ #
#   #      X #
#####
```

Figure 3: A possible optimal solution for instance *i01*.

Be careful with the solution output format ! Your solver must print the states with the exact same format as in Figure 3, with an empty line between each state. The labels “state [...]” are not part of the output, only the states with #, @, p and X. For this, do not modify the output printing code in the example file `pagecollect.py` and do not print anything in addition.



Questions

1. Model the PageCollect problem as a search problem; describe: (2 pts)
 - States
 - Initial state
 - Actions / Transition model
 - Goal test
 - Path cost function
2. Give an upper bound on the number of different states for a PageCollect problem with a map of size $n \times m$, with k pages to collect. Justify your answer precisely. (1 pt)
3. Give an admissible heuristic for a PageCollect instance with k pages. Prove that it is admissible. What is its complexity ? (2 pts)
4. **Implement** your solver. Extend the *Problem* class and implement the necessary methods and other class(es) if necessary. (1 pt)
5. **Experiment**, compare and analyze informed (*astar_search*) and uninformed (*breadth_first_graph_search*) graph search of aim-python3 on the 10 instances of PageCollect provided. (4 pts for the whole question)

Report in a table the time, the number of explored nodes and the number of steps to reach the solution.

Are the number of explored nodes always smaller with *astar_search*? What about the computation time? Why?

When no solution can be found by a strategy in a reasonable time (say 3 min), indicate the reason (time-out and/or exceeded the memory).

6. **Submit** your program on INGIInious, using the A^* algorithm with your best heuristic(s). Your file must be named *pagecollect.py*. Your program must print to the standard output a solution to the PageCollect instance given in argument, satisfying the described output format. (5 pts)