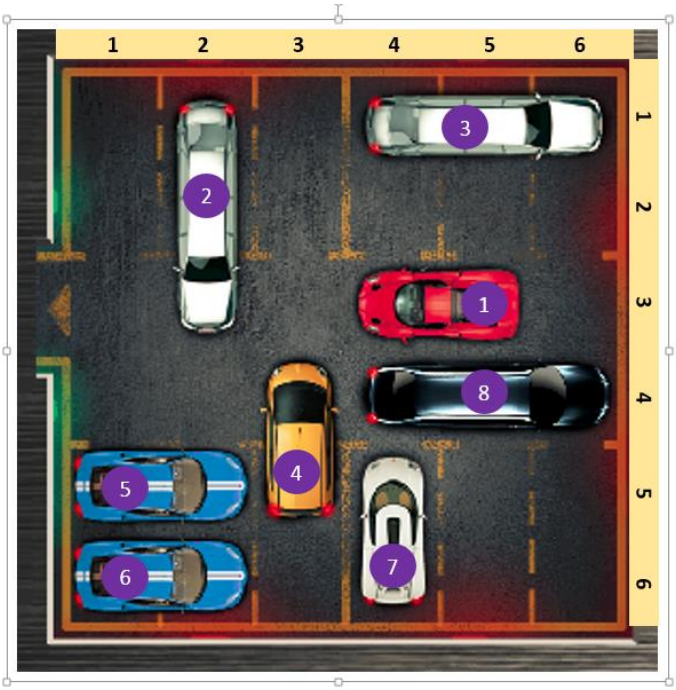
ASSIGNMENT 3: CONGESTED PARKING LOT

Goal: The goal of this assignment is to take a complex new problem and formulate and solve it using one of existing AI problem solvers. In this assignment we will use one of existing classical planners for solving a planning problem. Formulation as classical planning is a valuable way to approach a deterministic planning problem in the NP class. Over the years, the classical planners have become quite advanced and are often able to scale to decently sized real-world problems.

Scenario: Your car is stuck in a parking lot because of bad planning by the parking lot agent. You need to move other cars to get out of the lot. The goal is to figure out which cars to move and in what sequence to come out. A typical scenario is shown below in the figure:



The cars in the above figure are numbered 1 to 8. The car that we wish to get out of the lot will always be numbered 1. In this example, the parking lot can be considered in a 6x6 grid. The red car is occupying (4,3) and (5,3) and wishes to reach (1,3) to get out. The cars can only go forward and backward. That is car#1 can only move horizontally (in column#3) whereas car #2 can only move vertically in (row#2).

Classical Planner: We will use the classical planner LAMA (http://www.fast-downward.org/ObtainingAndRunningFastDownward) to solve this problem. LAMA (which is now part of the FastDownward repository) expects a PDDL domain and problem description. You will be provided a

scenario in a different input representation. You need to write code that reads your input description and converts into a PDDL description that LAMA can take as input. We will then execute LAMA on the PDDL version and it will solve the problem. You need to write a second piece of code that reads the solution of LAMA and transforms it into our output format. Overall, it is important to note that you will NOT write code to solve the problem. You will write code to transform the input representation of the problem into PDDL and transform LAMA's solution into our output representation.

Input format:

The first line will represent the MxN grid – M is the number of rows and N is the number of columns.

The second line will represent the number of cars in the grid. Recall that the first car will be your car that needs to leave the lot.

Each subsequent line will represent the position and length of each car in serial order (starting from 1). The format will be car-id, length, x-left, y-top, horizontal/vertical flag.

Finally, the last line will represent the goal of car 1 in the format x-left, y-top. The goal will always be on one of the sides of the grid.

A representation for the figure above is as follows:

6 6

8

1 2 4 3 H

2 3 2 1 V

3 3 4 1 H

4 2 3 4 V

5 2 1 5 H

6 2 1 6 H

7 2 4 5 V

8 3 4 4 H

1 3

Output format:

The solution of the problem will be a sequence of steps. The first line will represent the total number of steps in the final plan. Each subsequent line will be one step. One step is defined as one legal movement for one car (for as many legal distance). The format will be car number, U/D/L/R, length

For example, suppose there are 14 steps to the goal and first three steps of the solution are: move car #4 one step down, move car #1 one step right, and move car #8 three steps left. Also, lets say at the penultimate step the red car is occupying (5,3) and (6,3) and the path is clear for it to reach the goal, the solution may looks something like:

14

4 D 1

1 R 1

8 L 3

..

.. (10 more steps) ..

..

1 L 4

If the problem is unsatisfiable output a -1. If car#1 was already at the goal, output a 0.

What is being provided?

We are providing a format checker checker.py. By running "python checker.py <input problem file> <output solution file>" will test whether the solution is accurate for the problem. It will only work for satisfiable cases.

Code

Your code must compile and run on our VMs. They run amd64 Linux version Ubuntu 12.04. You are already provided information on the compilers on those VMs. These configurations are similar to GCL

machines like 'Todi' (have a RAM of 16 GB). Please supply a compile.sh script. Also supply two shell scripts run1.sh, run2.sh:

1. Executing the command "./run1.sh problem" will take as input a file named problem.txt and produce two file problem-domain.pddl and problem.pddl – the input files for LAMA. You can assume that problem.txt exists in the present working directory. Note that run1.sh can be called with any string (and not just 'problem').
2. Executing the command "./run2.sh problem" will use the generated problem.txt, problem.plan (and any other temporary files produced by run1.sh) and produce a file problem.out in the output format described above. You can assume that problem.txt, problem.plan (and other temp files) exist in the present working directory.
3. The TA will execute your scripts as follows:
    a. ./run1.sh problem
    b. src/translate/translate.py problem-domain.pddl problem.pddl
       src/preprocess/preprocess < output.sas
       src/fast-downward.py output --search "astar(lmcut())" > problem.plan
    c. ./run2.sh problem

While we have not given an explicit time limit in the assignment, we may cut off your program if it takes an excruciatingly long amount of time, say more than an hour or so.

Useful resources

1. http://www.fast-downward.org/ObtainingAndRunningFastDownward: The LAMA page
2. http://users.cecs.anu.edu.au/~patrik/pddlman/writing.html
3. http://www.cs.toronto.edu/~sheila/2542/s14/A1/introtopddl2.pdf

What to submit?

1. Submit your code in a .zip file named in the format **<EntryNo>.zip.b64** If there are two members in your team it should be called <EntryNo1>_<EntryNo2>.zip.b64 Make sure that when we run unzip the following files are produced in the present working directory:

   compile.sh
   run1.sh
   run2.sh
   writeup.txt

   You will be penalized for any submissions that do not conform to this requirement.

We will run your code on a few sample problems and verify the ability of your code to find solutions within a cutoff limit. The cutoff limits will be problem dependent and your translation does not need to depend on the cutoff limit, therefore it is not part of the input format. Of course, better translations will scale better and will possibly get higher scores.

2. The writeup.txt should have two lines as follows
   First line should be just a number between 1 and 3. Number 1 means C++. Number 2 means Java and Number 3 means Python.
   Second line should mention names of all students you discussed/collaborated with (see guidelines on collaboration vs. cheating on the course home page). If you never discussed the assignment with anyone else say None.
   After these first two lines you are welcome to write something about your code, though this is not necessary.

Code verification before submission: Your submission will be auto-graded. This means that it is absolutely essential to make sure that your code follows the input/output specifications of the assignment. Failure to follow any instruction will incur significant penalty. The details of code verification will be shared on Piazza (similar to A1).

Evaluation Criteria

1. Final evaluation on a set of similar problems. The points awarded will be your normalized performance relative to other groups in the class. In particular, the credit will be for solving the problem and computation time.
2. Extra credit may be awarded to standout performers. Standout performers may be those whose solution lengths are smaller than others, or who take substantially less time than others for computing the solutions.

What is allowed? What is not?

1. You may work in teams of two or by yourself. We do not expect a different quality of assignment for 2 people teams. At the same time, please spare us the details in case your team cannot function smoothly. Our recommendation: this assignment is rather easy. If you are having trouble finding partners, don't worry – you can do this assignment yourselves without much difficulty.
2. You can use any language from C++, Java or Python for translation into and out of PDDL as long as it works on our test machines. We will NOT be responsible for differences in versions leading to execution failures.

3. You must not discuss this assignment with anyone outside the class. **Make sure you mention the names in your write-up in case you discuss with anyone from within the class outside your team.** Please read academic integrity guidelines on the course home page and follow them carefully.
4. Please do not search the Web for solutions to the problem.
5. Your code will be automatically evaluated against another set of benchmark problems. You get a zero if your output is not automatically parsable.
6. We will run plagiarism detection software. Any team found guilty will be awarded a suitable penalty as per IIT rules.