Assignment 1, CSC384, Fall 2014

Out: October 1st

Due: October 15th, 10pm (Electronic Submission)

This assignment is worth 14% of your final grade

Be sure to include your name and student number as a comment in all submitted documents.

1 Introduction

1.1 Overview

In this assignment you will be provided with an A^* search solver. It will be your task to complete the code for two problem specifications as well as writing heuristics for these problems.

All work is to be done by individual students. You may confer with other students, but you should not share your code with other students or view other students code

1.2 Files Supplied

All provided files use Python 3. To make use of python 3 on the cdf machines, use the command `python3` (invoking `python` will default to Python 2.7).

The following four files make up the A^* search solver you are provided:

`open.py`
`cyclecheck.py`
`search.py`
`problem.py`

It is to your benefit to familiarize yourself with these files, but no alterations to them should be made for this assignment. `search.py` forms the basis of the search engine. Note that there is code for a variety of different searches (breadth first, depth first, IDA^*), not just A^*. `problem.py` provides abstract base classes for problems and search states that should be inherited by code for a problem description and `open.py` contains code related to the management of the open list. `cyclecheck.py` has code for cycle checking during search operation.

An example of a complete problem description is given in `waterjugs.py`, a simple problem mentioned in class. The goal is to use only the jugs provided for measurement in order to put the desired quantity of water in both jugs.

You will have to complete the problem descriptions in the following two files:

`fuelmaze.py`
`taskscheduling.py`

These correspond to the problems described in questions 1 and 2.

1.3 Submission

All submissions are to be electronic. You should submit the following three files via cdf:

`fuelmaze.py`
`taskscheduling.py`
`a1_answers.pdf`

2 Question 1: The Fuel Maze Domain ([40/100] marks)

2.1 Problem Description

In this question you will guide a robot with limited fuel over a grid filled with obstacles. The robot may be moved in one of four cardinal directions ("up", "down", "left", and "right") at the cost of one unit of fuel. Obstacles in

the grid prevent the robot from moving. Refuelling stations present on the grid allow the robot to refuel to its maximum fuel capacity.

Each grid has a fixed height and width. The robot has a fixed maximum fuel capacity. These will differ over distinct problem instances. Grids come with a set of fuel stations and a set of obstacles. You may assume that fuel stations and obstacles do not overlap.

The search should create a sequence of actions which guide the robot from its initial location to a goal location. The available actions are:

Right
Left
Up
Down
Refuel

Moving requires that there is an obstacle free space in the direction the robot wants to move to and that the robot has at least one unit of fuel remaining. Refuelling requires that the robot be at a location with a fuel station, after which the robot's fuel is increased to its maximum capacity. Assume all actions have a cost of one (we want to complete the task using a minimum number of actions).

2.2 Example

A new problem will be created with a constructor in the following format:

```
Fuelmaze(width, height, maximumfuelcapacity,
                [(8,6)], #a list of fuel stations given by (x,y) coordinate tuples
                [(3,0), (3,1), (2,2), (1,3), (1,6), (1,7), (0,8), (2,5), (3,4), (4,3),
                 (5,2), (5,1), (6,1), (6,4), (6,5), (7,6), (5,5), (5,7), (8,7)], # a list of obstacles
                (0,0),  #the initial position
                25,     #the initial fuel remaining
                (8,8))  #the goal position
```

Therefore the following constructor:

```
Fuelmaze(9, 9, 25,
                [(8,6)],
                [(3,0), (3,1), (2,2), (1,3), (1,6), (1,7), (0,8), (2,5), (3,4), (4,3),
                 (5,2), (5,1), (6,1), (6,4), (6,5), (7,6), (5,5), (5,7), (8,7)],
                (0,0),
                25,
                (8,8))
```

Would provide an environment that looks like this:

```
I..X.....
...X.XX..
..X..X...
.X..X....
...X..X..
..X..XX..
.X.....XF
.X...X..X
X.......G
```

Key: `X` is an obstacle, `F` is a refuelling station, `.` is open space, `I` is the initial position, and `G` is the goal. The robot will start with 25 fuel and has a maximum capacity of 25 fuel.

2.3 Work

In the file `fuelmaze.py`:
1) Implement the successor function `def successors(self, state):`
2) Implement the hash function `def hashable_state(self, state):`
3) Implement the goal checking function `def goal_check(self, state):`
4) Implement the Manhattan distance heuristic `def fuelmaze_h_manhattan(problem, state):`
5) Implement a heuristic of your own `def fuelmaze_h_custom(problem, state):`

Do not change the function header for any of the functions you implement!

As a written submission in `a1_answers.pdf` give short answers to the following questions:

1) Describe in your own words how your heuristic works. Is your heuristic admissible? Why or why not?

2) Compare your heuristic with the Manhattan distance heuristic and the uniform heuristics on several test cases on nodes explored and time taken to solve.

3 Question 2: The Task Scheduling Domain ([60/100] marks)

3.1 Problem Description

In this problem you will schedule jobs on a machine with multiple configurations. Each job has a configuration that is required to run that job. There is a time cost associated with changing configurations.

Every job has an earliest completion time, the earliest possible time at which the job can be completed. Jobs also have durations, how much time it takes to finish the job once started.

Each job also has completion deadlines. All completion deadlines have a cost, failing to meet a completion deadline (finishing after the deadline, meeting it exactly is fine) incurs this cost. For any given job only the largest cost is incurred, costs for failed deadlines are not cumulative. The goal is to finish all jobs, ideally with minimon cost incurred.

The valid actions for this problem are:

Set configuration('configurationname')
Perform Task('jobname')

3.2 Example

The following constructor call will create a task scheduling problem instance:

```
TaskScheduling(
    [('c1', 2), ('c2', 3), ('c3', 3), ('c4', 4) ],
    [
        ('j1','c1',3,3, [(8,10), (15,30)]),
        ('j2','c1',12,3,[(15,15), (20,30)]),
        ('j3','c2',5,3, [(12,15)]),
        ('j4','c2',15,3,[(15,15)]),
        ('j5','c3',6,3, [(10,5)]),
        ('j6','c4',6,3, [(10,10)])
        ],
    'c1'
)
```

This defines a problem with four configurations (`c1`, `c2`, `c3`, and `c4`) with setup times of 2, 3, 3, and 4 respectively. There are six jobs (`j1`, `j2`, `j3`, `j4`, `j5`, and `j6`). Each job has a specified earliest start time and duration. In this instance job `j2` may not be completed earlier than time 12. All jobs in this example have duration 3.

The final list associated with each job consists of tuples (time, cost) in which if the jobs is finished after the specified time, the cost is incurred. If the jobs is completed right at the deadline, that cost is not incurred.

The starting configuration in this example is `c1`

3.3 Questions

In the file `taskscheduling.py`:

1) Implement the state constructor `def __init__(self):`
2) Implement the string representation of a state `def __repr__(self):`
3) Implement the problem constructor `def __init__(self, configurations, jobs, starting_configuration):`
4) Implement the successor function `def successors(self, state):`
5) Implement the hash function `def hashable_state(self, state):`
6) Implement the goal checking function `def goal_check(self, state):`
7) Implement a heuristic of your own `def fuelmaze_h_custom(problem, state):`

Important note: For this problem you are required to create an admissible heuristic.

Do not change the function header for any of the functions you implement!

As a written submission in `a1_answers.pdf` give short answers to the following questions:

1) Describe in your own words how your heuristic works.

2) Informally prove the admissibility of your heuristic.