ECS 170: Spring 2020
Homework Assignment 2

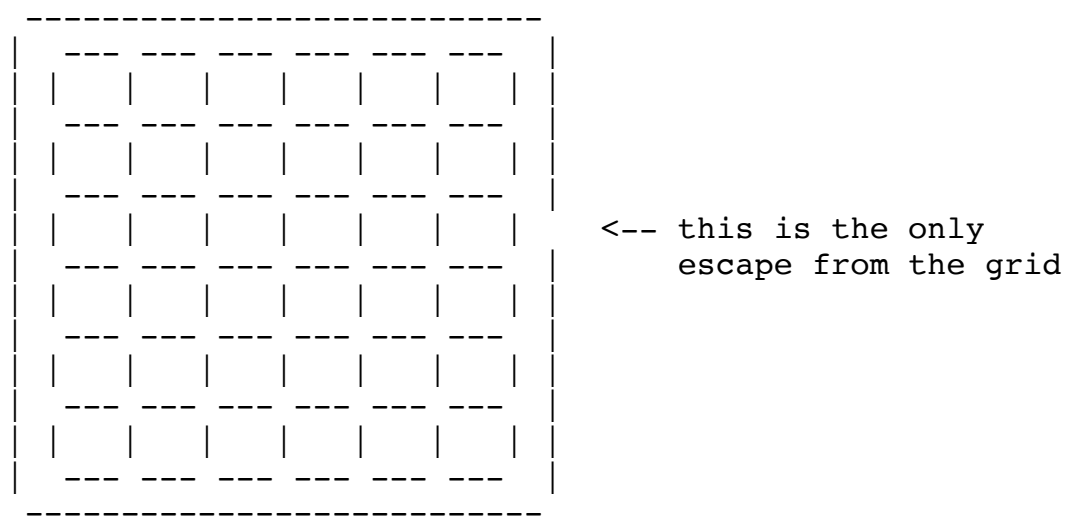
Due Date:

No later than Sunday, May 3, 11:59pm PDT. Note that your next homework assignment may be posted before then. You are expected to do this assignment on your own, not with another person.

The assignment:

The puzzle called "Rush Hour" (c. ThinkFun, Inc., www.thinkfun.com) begins with a six-by-six grid of squares. Little plastic cars and trucks are distributed across the grid in such a way as to prevent one special car from moving off the grid, thereby escaping the traffic jam. The initial configuration of cars and trucks is preordained by a set of cards, each containing a different starting pattern. A player draws a card from the deck of cards, sets up the cars and trucks according to the pattern, and then tries to move the special car off the grid by sliding the cars and trucks up and down or left and right, depending on the starting orientation of the cars and trucks.

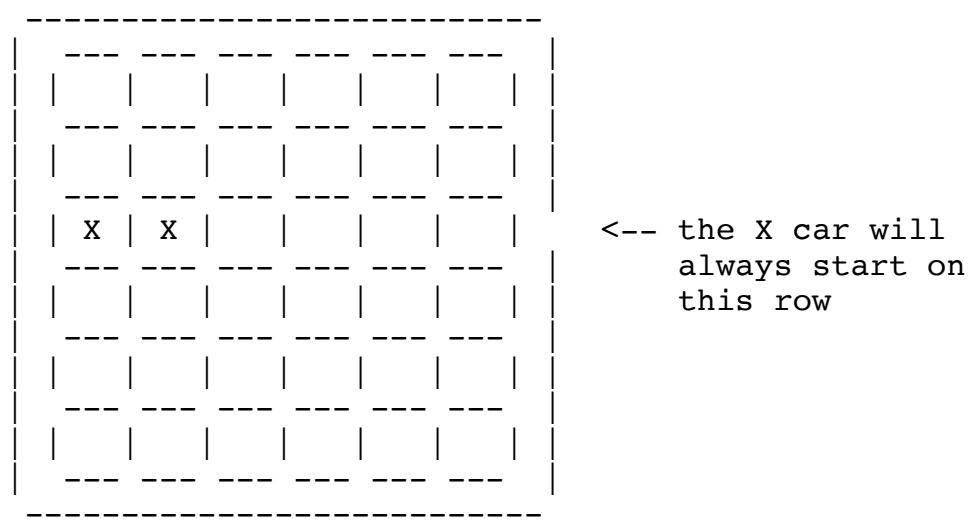
The grid looks something like this:



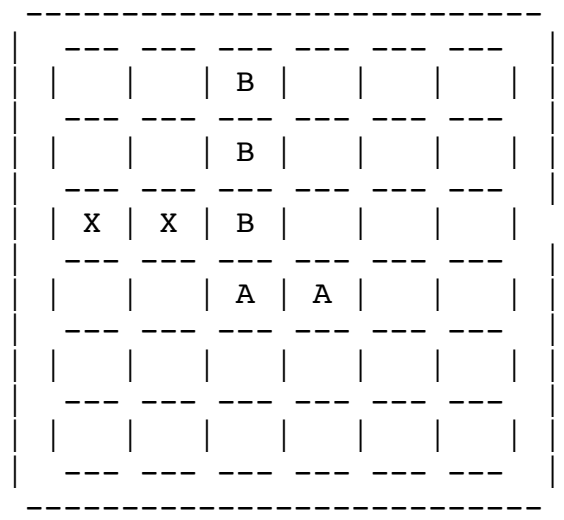
Cars occupy two contiguous squares, while trucks occupy three contiguous squares. All vehicles are oriented horizontally or vertically, never on the diagonal. Vehicles that are oriented horizontally may only move horizontally; vehicles that are oriented vertically may only move vertically. Vehicles may not be lifted from the grid, and there is never any diagonal movement. Vehicles may not be partially on the grid and partially off the grid.

The special car is called the X car. We'll represent it on the grid as two contiguous squares with the letter X in them. (We'll represent all other cars and

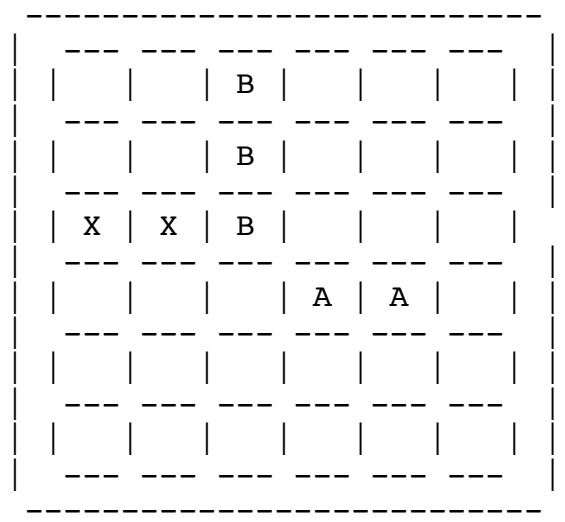
trucks with different letters of the alphabet, but the special car will always be labeled with the letter X.) The X car is always oriented horizontally on the third row from the top; otherwise, the X car could never escape. If it were the only car on the grid, it might look like this (although it could start anywhere on that third row):



Let's put a blocking car and a blocking truck on the grid now too. We'll label the car as A and the truck as B:

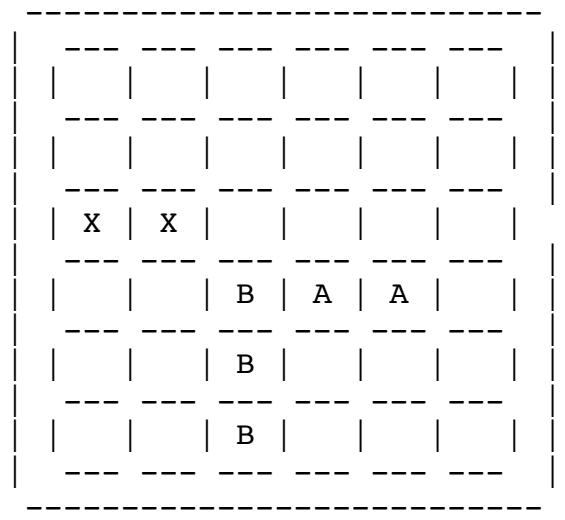


Car A is oriented horizontally on the fourth row, and it can slide only left or right. Truck B is oriented vertically in the third column from the left, and it can slide only up or down. If this configuration were the starting state for a puzzle, it would be pretty simple for us to solve it. To slide the X car all the way to the right and off the grid, we'd have to move the B truck out of the way. But to move the B truck, we'll first have to move the A car out of the B truck's path. So the first solution step might be to move the A car to the right by one square:

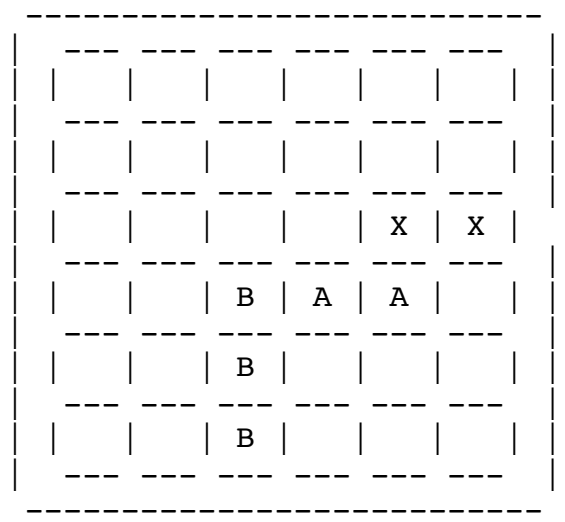


Note that moving the A car further to the right works, as does moving it two squares to the left, but *the best solution would move the X car off the grid in the fewest moves possible*. When any car or truck is moved one square in any direction, that adds one to the total number of moves. If a car or truck is moved one square in one direction, then later is moved back one square to where it started, that counts as two moves.

The next step would be to slide the B truck down three squares:

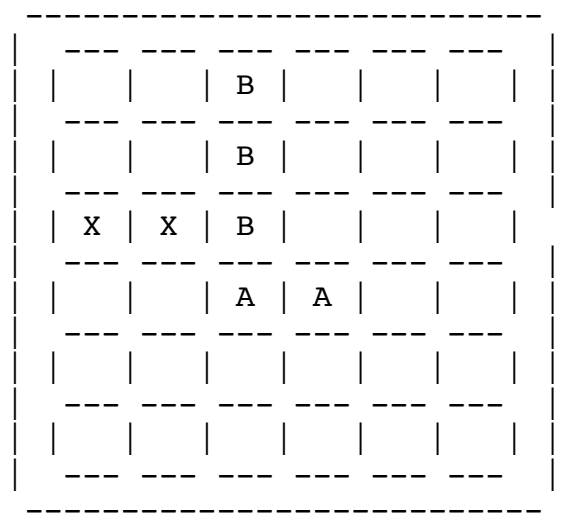


Now it's possible to move the X car all the way to the right and off the grid. It's not necessary to move it off the grid though; it's sufficient to move the X car to cover the rightmost two squares of the third row from the top:



Now we've solved the puzzle. Let's take another look at solving this "Rush Hour" puzzle, but this time from the perspective of a best-first search. The initial state in the state space may be any configuration of cars and trucks such that the X car is always oriented horizontally on the third row from the top. The goal state is any configuration of cars and trucks that can be obtained through correct application of the operators, such that the X car covers the rightmost two squares of the third row from the top. The operators are simply these: move a horizontally-oriented vehicle one square to the left, move a horizontally-oriented vehicle one square to the right, move a vertically-oriented vehicle one square up, and move a vertically-oriented vehicle one square down.

Your assignment is to use Python to write a program called `rushhour` which employs best-first search with the A* algorithm to solve a "Rush Hour" puzzle with any legal initial state. Your `rushhour` program expects two arguments. The second argument is a description of the initial-state as a list of six strings, each string containing six characters. For this initial configuration:



the list of strings passed to `rushhour` would look like this:

```
["--B---","--B---","XXB---","--AA--","------","------"]
```

The first string corresponds to the top row, the second string corresponds to the next row down, and so on. If this list were formatted nicely, it would look more like the grid above:

```
 ["--B---",
  "--B---",
  "XXB---",
  "--AA--",
  "------",
  "------"]
```

We don't need to pass the goal state, as it will always be the same regardless of the initial state. The goal state will always be a legally-obtained configuration of cars and trucks with the X car on the rightmost two squares of the third row.

When (if?) the goal has been reached, `rushhour` should terminate and print a list of successive states that constitutes the path from the initial state to the goal state. So, if your TA invokes `rushhour` in this way (we'll explain that integer in the first argument later):

```
>>> rushhour(0, ["--B---","--B---","XXB---","--AA--",
"------","------"])
```

your program should print a solution like this:

```
   --B---
   --B---
   XXB---
   --AA--
   ------
   ------

   --B---
   --B---
   XXB---
   ---AA-
   ------
   ------
```

```
------
--B---
XXB---
--BAA-
------
------

------
------
XXB---
--BAA-
--B---
------

------
------
XX----
--BAA-
--B---
--B---

------
------
-XX---
--BAA-
--B---
--B---

------
------
--XX--
--BAA-
--B---
--B---

------
------
---XX-
--BAA-
--B---
--B---

------
------
----XX
--BAA-
--B---
--B---
```

```
Total moves: 8
Total states explored: 37
```

OK, I made up that last number. But I'll explain later what your program should put there. One more thing to be explained later.

The result above reflects the fact that you'll be using operators that move vehicles only one square in any direction at a time. Do not employ additional operators that move a vehicle two, three, or four squares at a time. Keep it simple.

If you wish, you can convert my representation for the "Rush Hour" puzzle into any other representation of your choice if you think that would make your life easier. I make no claim that the representation I'm using above for passing the start state to your program is necessarily a great representation to use in solving the problem. It just happens to be convenient for representing the start state. So like I said, if you think things might go easier for you if you use a different representation, please feel free to do so. However, you still have to deal with my representation as input to your function as specified above.

In your program, we want to see lots of well-abstracted, well-named, cohesive, and readable functions. Comments are required at the beginning of every function and must provide a brief description of what the function does, what arguments are expected by the function, and what is returned by the function. Additional comments that help the TA who grades your program understand how your program works may make the TA happy, and happy graders are generous graders.

What about the heuristics? You must implement two different h(n) functions. One is called the blocking heuristic, in which h(n) returns zero for the goal state, or one plus the number of cars or trucks blocking the path to the exit for all other states. For the sample state given at the beginning of this document, there is one truck blocking the path to the exit, so h(n) in this case would be two. To activate this heuristic, the first argument in the call to `rushhour` will be a 0. (See, I told you we'd explain that first argument.)

The other heuristic is one of your own design. You should try for a heuristic that is at least as effective as the blocking heuristic. A trivial heuristic will not get credit. To activate this heuristic, the first argument in the call to `rushhour` will be a 1.

Speaking of things to be explained later, here's what we want your program to print after it has printed the solution to the puzzle:
'

```
Total moves: 8
Total states explored: 37
```

The first value should be obvious - it's the number of moves used in the solution. The second value is the number of times a node is removed from the front of the

frontier and examined (see the algorithm in Episode 6). This gives us an idea of how much work is being done to find the solution. One would hope that better heuristics would result in smaller values here.

Additional Notes:

- The X car (the one to be moved off the grid) will always be indicated by the letter X, and will always be somewhere on the third row.

- The number of other cars and trucks on the grid is limited only by the available space on the grid. Do not assume that there will be only one other car and one other truck just because that's what's shown in the example.

- The other cars and trucks will be labeled with single letters from the alphabet, but don't assume that those letters will always be taken from the beginning of the alphabet.

- Implement best first search with the A* algorithm. Do not implement something else, even if you think you have a better way to do it.

- Comments in your code should make it easy for the TAs to find your best-first search code and your two heuristics. You should use comments to explain how your newly-created heuristic works and why you think it might be better than the blocking heuristic. If the TAs can't find what they will be looking for, you will not get the grade that you are looking for.

- There are many resources on the Internet and elsewhere that might help you with this assignment. We've seen them, and we know some of you will look for them. (Yes, it's true...the CS department even has its own Chegg account.) I strongly advise you to read the UC Davis Code of Academic Conduct and the syllabus for ECS 170 so that you thoroughly understand what behavior is and is not acceptable. I strongly advise against looking at any code you might find, because if it ends up in your submission, what I've intended to be a fun course for you will turn into something remarkably un-fun for you in a hurry.

- I have tried to anticipate every possible question, but I know that's impossible. You will come up with things that I didn't anticipate, or you'll find inconsistencies or ambiguities, and I may have to make adjustments to the assignment as the days go by. Once again, please try to be flexible.