B551 Assignment 0: N-queens and Python

Fall 2018
Due: Sunday September 9, 11:59:59PM Eastern (New York) time
(You may submit up to 48 hours late for a 10% penalty.)

This assignment will give you practice with posing AI problems as search and an opportunity to dust off your coding skills. Because it's important for everyone to get up-to-speed on Python, for this particular assignment you must work individually. Future assignments will allow you to work in groups. Please read the instructions below carefully; we cannot accept any submissions that do not follow the instructions given here. Most importantly: please **start early,** and ask questions on Piazza or in office hours.

If you don't know Python, never fear – this is your chance to learn! But you'll have to spend some significant amount of out of class time to do it. Students in past years have recommended the Python CodeAcademy website, `http://www.codeacademy.com/learn/python`, and Google's Python Class, `https://developers.google.com/edu/python/`. There are also a wide variety of tutorials available online. If you're already proficient in one particular language, you might look for tutorials with names like "How to code in Python: A guide for C# programmers," "Python for Java programmers," etc. Feel free to share any particularly good or bad resources on Slack and/or Piazza. The instructors are also happy to help during office hours. **If you are struggling, please come and get help!** We can't help if you don't ask.

Academic integrity. We take academic integrity very seriously and, to maintain fairness to all students in the class and integrity of our grading system, we will prosecute any academic integrity violations that we discover. *Before beginning this assignment, make sure you are familiar with the Academic Integrity policy of the course, as stated in the Syllabus, and ask us about any doubts or questions you may have.* To briefly summarize, you may discuss the assignment with other people at a high level, e.g. discussing general strategies to solve the problem, talking about Python syntax and features, etc. You may also consult printed and/or online references, including books, tutorials, etc., but you must cite these materials (e.g. in source code comments). We expect that you'll write your own code and not copy anything from anyone else, including online resources. *However, if you do copy something (e.g., a small bit of code that you think is particularly clever), you have to make it explicitly clear which parts were copied and which parts were your own. You can do this by putting a very detailed comment in your code, marking the line above which the copying began, and the line below which the copying ended, and a reference to the source.* Any code that is not marked in this way must be your own, which you personally designed and wrote. You may not share written answers or code with any other students, nor may you possess code written by another student, either in whole or in part, regardless of format.

Introduction

Imagine that you have an empty chessboard of size N – i.e., an $N \times N$ grid of squares. Our goal is to find a way of placing N queens on the board such that no two queens share the same row, column, or diagonal – in other words, so that no queen could "take" any other queen. This is called the *N-queens problem.* For example, when $N = 1$, there's a very easy solution (the queen goes on the only square!). For $N = 2$, there's no solution at all, nor is there a solution for $N = 3$. But there are solutions for larger boards, including the standard chessboard with $N = 8$.

But let's first start with a related, simpler problem: N-rooks. N-rooks is like N-queens except that the goal is to place N rooks so that no two of them are in the same row or column. This is a lot easier because there are many more possible solutions, since we don't need to worry about diagonals. (For example, there *is* a solution for $N = 2$ and $N = 3$.)

How do we pose this as search? We'll need to define the five parts of our abstraction, and there are various ways of doing this. To see this, think about how you might solve the problem as a human. You could start with an empty board and try to add one piece at a time, each time trying to add the piece such that it does not conflict with any existing pieces. Or you could randomly arrange N rooks on the board and in each step,

move one of the pieces to a vacant spot until none of the pieces conflict with one another. You could even fill the board with N^2 pieces and then remove one at a time until you are left with N that do not conflict. Each of these strategies would probably work, and each one involves defining the abstraction (state space, start state, goal state, successor, cost function) in a different way.

To get you started, we've created a partial solution to the N-rooks problem that uses one particular abstraction and one particular algorithm. Our code is available via Canvas. You can specify a value of N you're interested in, and then it will try to find a solution and print out the first one it finds. The problem is that it's really slow. You can easily test that it works for $N = 1$ and $N = 2$, but larger values of N seem to take a really long time, or never seem to finish at all.

You can run it like this (on the SOIC Linux machines):

```
./nrooks.py N
```

where N is the number of rooks you're interested in. *Note that if you upload the code from a Windows machine, you may need to change the line endings like this:*

```
dos2unix nrooks.py
```

You will likely also have to change the Unix permissions on the nrooks.py file to tell the operating system that it is a program, like this:

```
chmod u+x nrooks.py
```

What to do

1. Spend some time familiarizing yourself with the code. Write down the precise abstraction that the program is using and include it in your report. In other words, what is the set of valid states, the successor function, the cost function, the goal state definition, and the initial state?

2. You've probably noticed that the code given is implemented by depth first search (DFS). Modify the code to switch to BFS instead. What happens when you run the code for different values of N now, and why?

3. Recall from class that there are usually many ways to define a state space and successor function, and some are more practical than others (e.g. some have much larger search spaces). The successor function in the code is defined in a very simplistic way, including generating states that have N+1 rooks on them, and allowing "moves" that involve not adding a rook at all. Create a new successors() function called successors2() that fixes these two problems. Now does the choice of BFS or DFS matter? Why or why not?

4. Even with the modifications so far, N=8 is still very slow. Instead of allowing the successor function to place a piece anywhere on the board, let's define a successor that is much more orderly: it's only allowed to add a piece to the *leftmost column of the board that is currently empty.* (For example, if the board is empty, the rook has to be placed in the first column; if there are 4 rooks on the board, the next one has to go in the fifth column, etc.) Modify the code to implement this alternative abstraction with a successors function called successors3(). Feel free to make other code improvements as well. What is the largest value of N your new version can run on within about 1 minute?

   Tip: In Linux, you can use the `timeout` command to kill a program after a specified time period:

   ```
   timeout 1m ./nrooks.py N
   ```

5. Now, create a new program, `a0.py`, with several additional features, including the ability to solve both the N-rooks and N-queens problems, and the ability to solve these problems when some of the squares of the board are not available (meaning that you cannot place a piece on them). Your program must

accept at least 3 arguments. The first one is either `nrook` or `nqueen`, which signifies the problem type. The second is the number of rooks or queens (i.e., N). The third argument represents the number of unavailable positions. The remaining arguments encode which positions are unavailable, using row-column coordinates and assuming a coordinate system where (1,1) is at the top-left of the board. Taking the 7-rooks problem with position (1, 1) unavailable as an example, we might run:

```
[<>djcran@tank ~]$ ,.a0.py nrook 7 1 1 1
```

which means that one square is unavailable, and it is at row 1 and column 1. One possible result could be:

```
[<>djcran@tank ~]$ ./a0.py nrook 7 1 1 1
X R _ _ _ _ _
R _ _ _ _ _ _
_ _ R _ _ _ _
_ _ _ R _ _ _
_ _ _ _ R _ _
_ _ _ _ _ R _
_ _ _ _ _ _ R
```

where `R` indicates the position of a rook, underscore marks an empty square, and `X` shows the unavailable position. Or for the 8-queens problem with (1, 2) and (1,8) unavailable, one possible result could be:

```
[<>djcran@tank ~]$ ./a0.py nqueen 8 2 1 2 1 8
_ X _ _ Q _ _ X
_ _ _ _ _ _ Q _
_ Q _ _ _ _ _ _
_ _ _ _ _ Q _ _
_ _ Q _ _ _ _ _
Q _ _ _ _ _ _ _
_ _ _ Q _ _ _ _
_ _ _ _ _ _ _ Q
```

where the `Q`'s indicate the positions of queens on the board. As a special case, there can be 0 unavailable squares, in which case only exactly three arguments are given to a0.py. Please print only the solution in exactly the above format **and nothing else**. The output format is important because we will use an auto-grading script to test and grade your code.

Hints and tips

Output format instruction. In the assignments in this class, we will usually give you specifications on the format of the input and output to your program, and you must be very careful to satisfy these specifications. This is for two reasons: (1) we typically test your code with a semi-automatic grading script that expects input and output in a particular format, and (2) an important part of becoming a better programmer is to learn how to work within a client's precise specifications. Make sure your code exactly satisfies the input and output specifications given above.

Testing your code. It's very important that you test your code on the CS Linux servers, `burrow.soic.indiana.edu` (aka silo), before you submit it. Because of differences in Python versions, operating systems, etc., your code may work beautifully on your home computer but fail to run on the CS Linux servers, which will likely lead to a lower grade on the assignment than you deserve. So please make sure to test your code on the CS servers well ahead of time, so that you can debug and address any problems that might arise.

To help you verify that your code satisfies our specifications, we will prepare a simple test program which we will release soon. The test program will run your program and indicate whether it can understand the output. Note that it will not verify that your solution is correct, just that the output format is correct.

Python versions. Unfortunately, there are two major versions of Python in common use, Python 2.7 and 3.5. These versions are very similar but incompatible enough that many programs written for one version will not work with the other. Despite that Python 3 is now almost 10 years old, Python 2.7 is still widely supported. You can use either version, but you should set the first line of your program – the so-called "hash-bang" or "she-bang" line – to specify the right version, either:

```
#!/usr/bin/env python3
```

or

```
#!/usr/bin/env python2
```

and make sure to test on the SOIC Linux machines. If you don't do this, we are likely to get syntax errors when we run your code, and the grader will likely assume that your code does not work at all.

There are many differences between the two versions of Python (see `https://docs.python.org/3/whatsnew/3.0.html` for a comprehensive explanation), but two come up particularly often. First, in Python 3, *print* is a function, so its arguments must be surrounded by parentheses. Second, in Python 3, dividing one integer by another integer performs floating point division, instead of integer division.

Grading. Your code should be correct and clear. Please use comments and meaningful variable names so that we can understand it.

Again: it is *very* important that your program satisfies the requirements given above on input format, output format, Python version, etc. It is thus **crucial** that you test your code on the SOIC Linux servers. We will take off points if we cannot run your code, or if we have to modify it in any way to get it to run on the SOIC Linux systems.

Extra credit. For a little extra credit, implement the N-knights problem, which should be invoked using *nknight* as the first parameter to a0.py. The goal of this puzzle is to place N knights on the board such that none of them can take any of the others, where again some squares are unavailable.

What to turn in

Turn in three files via Canvas: (1) a PDF or text document that answers the questions in 1, 2, 3 and 4 above, (2) your modified version of nrooks.py, and (3) your source code for a0.py. **Again** (for the third time!), make sure that: (1) your code runs correctly on the CS Linux servers, `burrow.soic.indiana.edu` (aka silo), since this is where we'll test your code; (2) your code has the correct file name and can be run in the required format. *We will deduct points if we have to modify your code in order to make it run on our system.* (Every year, some students ignore this, which both greatly increases grading time and greatly decreases their grade. :-)