CSE 150 Programming Assignment 1

Due April 15, 2016 11:59 PM

1 Overview

In this assignment you will be solving a problem based on prime numbers. In this problem, you will be given a initial prime number and a final prime number. Your task is to reach the final one through a sequence of intermediate prime numbers which are generated by changing one digit at a time. For example if your initial prime number is 103 and final prime number is 199, then one such possible path is
$103 -> 109 -> 199$

We will be using various search algorithms in this assignment.
Note - We are changing one digit at a time here in the current prime number to generate another prime number. For example from 103, we can reach following prime numbers - 107,109,113,173,193 and 503 by changing one digit at a time.

Moreover in the above problem there can be multiple paths to reach 199 from 103, for example
$103 -> 107 -> 109 -> 139 ->199$
$103 -> 109 -> 139 ->199$
$103 -> 107 -> 137 -> 139 ->199$
$103 -> 109 ->199$ and so on.

Note- Wherever, there is a conflict, i.e. there are multiple children you could process at next step, you can process the children in any order. All solutions are equally valid.

2 Problems

Write the solution to each problem in **separate files** with following **naming convention**(assignment1_p1.py, assignment1_p2.py, etc.). The details around how to submit the assignment will be shared later. Also, write the email addresses and PID of your group members in the author metadata, separated by commas. (See the sample code below).

Testing

Except for the last problem, your code will be run on several different sets of initial and final prime numbers, including but not limited to the ones given as examples. The test examples are in the form of space separated values, with each line marking a different test case in the puzzle. So, your code may look like the following:

assignment1_p1.py

```
__author__ = 'student1@ucsd.edu,student1PID,student2@ucsd.edu,
    student2PID,student3@ucsd.edu,student3PID'

def getPossibleActions(currentPrime):
        # This method would return the list of prime
        # numbers reachable from current prime.
        # Note - this should not include the prime numbers
        # which have already been  processed, either in the
        # frontier or in the closed list.
        return listOfPrimes

def getPath(startingPrime,finalPrime):
        # your code here
        return path

def main():
        primes=str(sys.stdin.readline()).split()
        print(getPath(primes[0],primes[1]))

if __name__ == '__main__':
    main()
```

Your own testing should be thorough enough that there are no surprises here. You can write in either Python 2 series or 3 series, but try not to use extra packages. Obviously, don't do anything other than what is needed for the assignment (especially importing the os package, which we will obviously check.)

2.1 Problem 1

Write a breadth first solver for the puzzle. Your program should print the sequence of intermediate prime numbers used to reach the solution. Submit the solution to this problem with the name assignment1_p1.py

Note:

- If the puzzle is not solvable, print UNSOLVABLE.
- Use the Graph-Search algorithm to search; i.e. don't visit the same node more than once. See Figure 3.7 in Russell and Norvig.

Examples

input.txt

```
103 199
```

output.txt

```
103 109 199
```

Again, we don't care how you get to the solution, as long as it is optimal, in the being the shortest possible path. Here, the answer could have been 103,193 and 199

input.txt

```
103 1231
```

output.txt

```
UNSOLVABLE
```

As there can be no such path from a 3 digit prime number to a 4 digit prime number. Note that this is just an example, a completely uninformed search algorithm would search the entire tree before returning unsolvable, so please don't insert any other checks in the code such as whether two elements have same number of digits etc.

Note: you can test these examples by running the Python code as follows:
$ python assignment1_p1.py < input.txt

2.2 Problem 2

Do the same as **Problem 1**, but for the depth-limited search algorithm (depth-first search algorithm with a depth limit). The depth limit should be 5 for this problem (where the root node is "depth 0".) Submit the solution to this problem with the name assignment1_p2.py
Note:

- Use the Graph-Search algorithm to search; i.e. don't visit the same node more than once.
- Remember that it should print UNSOLVABLE if the puzzle is not solvable using this limit.

Examples

input.txt

```
103 199
```

output.txt

```
103 109 179 199
```

Note that there can be multiple possible solutions to this problem based on the order in which you process children. Here, the answer could have been 103,109 and 199.

2.3 Problem 3

Do the same as **Problem 2**, but for the iterative deepening depth-first algorithm. Use absolute depth limit of 8 for this problem, so that the code will run in reasonable time even for unsolvable cases. Submit the solution to this problem with the name assignment1_p3.py

Examples

input.txt

```
103 199
```

output.txt

```
103 109 179 199
```

Note here there are many more node visits than Problem 2, because (a) With the depth limit of 0, the initial node 103 is checked (1 visit), and (b) With the depth limit of 1, the initial node is checked (1 visit) and its children 109,107,113,173,193 and 503 (6 visit) and so on.

2.4 Problem 4

(Extra Credit) Do the same as **Problem 1**, but for the bidirectional search algorithm, outputting the path followed from both start and end element till the middle element in separate lines. Submit the solution to this problem with the name assignment1_p4.py

Examples

input.txt

```
103 199
```

output.txt

```
103 109
```

199 109

input.txt

103 269

output.txt

103 163 263
269 263 163

input.txt

103 1231

output.txt

UNSOLVABLE

2.5 Problem 5

a) Do the same as **Problem 1**, but implement the A* algorithm with hamming distance as the heuristic. Submit the solution to this problem with the name assignment1_p5.py b) Try to come up with your own heuristics(atleast one). Comment on change of behavior. Does your heuristic guarantee optimal behaviour?

2.6 Problem 6

Write a report on your assignment, with the following component written in well structured English sentences:

- Description of the problem and the algorithms used to solve problems 1 - 5.
- Describe the data structure used in each algorithm.
- Perform some analysis on the efficacy of the different algorithms with different puzzles. For instance, you can try this problem for large prime numbers and for different sets of prime numbers. You can then measure the number of nodes visited / maximum size of the queue / time it took to run the code/ path length given by algorithm etc., for each case. How do they compare against what you expect from the big-O analysis? How do they compare against each other?
- Results of the analysis and a short discussion. It should include **at least one graph with proper labels** that shows how the quantity you measured changes with what you varied.

- A paragraph from each author stating what their contribution was in this assignment.

You will be required to upload the PDF file of the report as well as code for all the problems. We will be testing your code on different test cases using a script, so please adhere to the input and output format specified in the problems. Again note that your code should run directly by running your python file for any input file like this -

$ python assignment1_p1.py < input.txt

Important Points-

- Start testing on small, trivial problems (such as the above examples).
- Try to optimize code-reusability. Once Problem 1 has been coded, implementing the later problem should be much easier. However, since each submission has to be self-contained in a single file, you'd have to copy-paste common routines (unfortunately.)