Module 1: Critical Thinking

Algorithms in Practice – Option #2

Nolan Byrnes

CSC506 – Design and Analysis of Algorithms

Colorado State University – Global Campus

Professor Isaac Gang

September 16th, 2022

Module 1: Critical Thinking

Merriam-Webster defines an algorithm as “a step-by-step procedure for solving a problem or accomplishing some end” (Merriam-Webster, 2022). Food recipes, instructions on how to drive a car, and instructions to come up with the next highest number in a digit string could all be algorithms, as they all include a set of instructions that help achieve an end goal. To come up with an algorithm, you must first know what the problem is, and the outcome that is desired. For example, when given an arrangement of digits zero through nine, we may want a way to determine what the next highest number is. The next step of designing the algorithm is to give step-by-step instructions on how to achieve this end goal. In this paper, we discuss how I came up with the algorithm to find the next highest number when given an arrangement of digits, the Python implementation, and the runtime complexity of the developed algorithm.

Algorithm Discovery

When given an arrangement of digits, we can design an algorithm to come up with the next highest number that is available. We do know that the highest number possible is one that is in decreasing order. After writing a list of numbers and arranging the digits to come up with the next highest number, I began to see a pattern. Keeping in mind that the highest arrangement of digits is in decreasing order, I looked at the numbers from right to left, and noticed that when I found a digit that was not in increasing order, I would switch that number with the next highest digit that was to the right of it, and then sort the numbers in increasing order that was to the right of the digit that was originally the number that was not in increasing order.

So, for the number 1243, the digits 3 and 4 are in increasing order, and 2 is the digit that breaks the increasing order pattern. By switching the 2 with the 3, and then sorting the digits 4 and 2, we come up with 1324. By applying this pattern multiple times, we come up with the next highest number, until we end up with the number 4321.

Python Implementation

In my python implementation of the algorithm shown in Figure 1 below, I first converted the string of digits into a list of integers. This is so that I could compare the integers to one another. The algorithm then loops through all the digits starting from the last index, to the first, identifying the first number that is not in increasing order. The algorithm then splits the digits into two lists, where the right list contains all the digits to the right of the digit that was not in increasing order. The algorithm then proceeds to identify the smallest number in the right sequence that is greater than the last digit in the left sequence by looping through the right list. The algorithm then switches the last index of the left list with the smallest number identified in the right list. After sorting the right list to be increasing order, the two lists are combined, converted back into a digit-string, and the digit-string and True is returned.

Figure 1.

Python implementation of Algorithm

Text

Description automatically generated

Note. This algorithm determines the next larger value that can be represented with a provided digit string .

Runtime Complexity

When we come up with an algorithm, we want to come up with a solution in the most efficient way possible. To determine how efficient an algorithm is, we can analyze its runtime complexity, which “is a function, T(n), that represents the number of constant time operations performed by the algorithm on an input of size N” (Lysecky, 2019). Since the runtime of the algorithm will vary based on the inputs, it is best to identify the best- and worst-case scenarios. In the best-case scenario, the digit string will already be in a decreasing order. If the digit string is in decreasing order, the algorithm will only need to iterate through all the digits in the digit string and return False, indicating that there is not a next number, so the time complexity will be T(n), *n* representing the number of digits in the digit string. The worst-case scenario would be for the digit string to not be in decreasing order. In the worst-case scenario, the algorithm must go through a total of 2 loops and a sort, one loop to identify the first digit that breaks the increasing order pattern from right to left, one loop to identify the smallest digit that is greater than the digit that broke the increasing order pattern, and then a sort to order the digits on the right. Python’s sort function uses the TimSort algorithm, which has a n\*log(n) complexity (GeeksforGeeks, 2021). The time complexity for worst-case scenario of the algorithm would be T(n\*log n), with *n* representing the two loops, and *log n* would represent the sort that is performed on the right digits.

Conclusion

Algorithms are a set of instructions to accomplish a given task. To come up with an algorithm, you must first understand the problem that you are trying to solve. Before coming up with an algorithm to determine what the next highest number in a digit string is, we had to understand the problem, identify any patterns, and recognize what makes the highest number in a digit string. It was helpful for me to write down the steps, as I was able to identify which numbers were being replaced and ordered when coming up with the next number. By writing down each step that I took when coming up with the next number in a digit string, I was able to translate the steps that I took into Python code. To determine how efficient an algorithm is, we could analyze its runtime complexity for its best- and worst-case scenarios. The best-case scenario for an algorithm is one in which the algorithm would have to take the least amount of steps to return the solution, whereas the worst case would be where the algorithm would have to take the most steps.

**REFERENCES**

GeeksforGeeks. (2021, June 26). TimSort. https://www.geeksforgeeks.org/timsort/

Lysecky, R., & Vahid, F. (2019). 1.6 Algorithm efficiency. In Design and Analysis of Algorithms. zyBook.

Merriam-Webster. (2022, September 12). Definition of algorithm. Dictionary by Merriam-Webster: America's most-trusted online dictionary. https://www.merriam-webster.com/dictionary/algorithm