CSC3310 — Program #3 — Dynamic Programming

Objectives

- Design a dynamic programming algorithm for a given computational problem
- Justify the correctness of an algorithm
- Perform asymptotic complexity analysis of the run time
- Design and execute benchmarks for an algorithm

Overview

For this assignment, you will apply dynamic programming to create an algorithm that solves the maximal substring problem.

The Problem

The problem that you will be solving is called the MAXIMALSUBSTRING problem.

> MAXIMALSUBSTRING
> **Input** A string a of n characters, and a string b of m characters
>
> **Output** A string s such that s is a substring of both a and b, and there is no string s' where $|s'| > |s|$ and s' is a substring of both a and b

For example, if we are given the strings "orange" and "banana" we would return "an", for "banana" and "apple", we return "a", and for "apple" and "pineapple", we return "apple". For "pinaeapple" and "orange" we could return either "e" or "a".

There are $\mathrm{O}\left(n^2\right)$ possible substrings in a and (similarly) $\mathrm{O}\left(m^2\right)$ in b. We can avoid some complexity by only checking substrings of the same length, but a brute-force algorithm will still check $\mathrm{O}\left(n^2 m\right)$ pairs.

N.B String operations are not constant time. Comparing two strings takes $\mathrm{O}\left(k\right)$ time, where k is the string length. Similarly, concatenation $x + y$ takes $\mathrm{O}\left(k_y\right)$ time (some implemetations require $k_x + k_y$; we will assume ours is not one such). Slicing (getting a substring) needs time proportional to the length of the substring.

Your goal is to create a faster algorithm using a dynamic-programming strategy. **Hint:** Consider substrings that start/end at a particular pair of characters.

Deliverables

Complete a lab report containing the following:

- A paragraph describing the approach you will use to solve the problem. Provide at least 2 illustrations that explain the approach.
- High-level pseudocode for an algorithm that uses that rule to solve the computational problem for any input
- An explanation and justification for why your algorithm is correct (1-3 paragraphs)
- A table of your test cases, the answers you expect, and the answers returned by your implementation of the algorithm.
- An analysis of the asymptotic runtime of your algorithm, using techniques from this class.
- A table and graph from benchmarking different lists with different sizes and values of m and n. Implement a brute-force solution and benchmark your implementation against it.

  If the benchmarks do not support your theoretically-derived run time and/or do not provide evidence that the run time of your algorithm grows more slowly than the brute-force approach, this may indicate a flaw in your implementation.
- An appendix containing all of your source code and test cases.

Submit your report to Canvas in PDF format.

In addition to the report, you should submit a zip archive with your source code.

Rubric

		Full Credit	Partial Credit	No Credit
In class participation	10%			
Lab report writing and presentation quality	10%			
Solution Approach Description	10%	Approach is well-described, with all necessary details to implement and/or to explain to someone else	Approach is decently described with most details necessary to implement. Explanation may be unclear in places	The description is insufficient to implement or explain to others.
Algorithm, as described in Pseudocode	10%	Algorithm is correct for all allowed inputs	Algorithm is correct for most inputs	Algorithm is incorrect for many inputs
Justification of Correctness	10%	Uses techniques described in class to provide a solid and convincing argument that the algorithm is correct	Provides a somewhat convincing argument that the algorithm is correct	Argument contains one or more serious flaws
Asymptotic run-time analysis	10%	Analysis is correct for the provided pseudocode	Analysis contains minor flaws	Analysis is significantly flawed
Algorithm Implementation	10%	Implementation is both faithful to the pseudocode and correct	Implementation is mostly faithful to the pseudocode and/or correct for most inputs	Implementation is not faithful to the pseudocode or not correct for common inputs
Test Cases	10%	Test cases consider a range of problem sizes, complexities, classes, and edge cases	Limited number of test cases only testing obvious or simple cases	Only the examples
Benchmarking	10%	Benchmark experiments were set up and implemented correctly.	Benchmark experiments, implementations, or results are mostly correct.	Benchmark experiments, implementations, or results are flawed.
Algorithmic Runtime	10%	Runtime is asymptotically $\mathrm{O}\left(n \times m\right)$, as determined by both empirical results and theoretical analysis		Runtime is asymptotically equal-to or slower-than $\mathrm{O}\left(n^3 m\right)$ brute-force search