CSCI 4118/6105 Lab 2 (Part 1):

Dynamic Programming

Winter 2022

Objective

1. Understand the time and space complexities of Algorithms that use Dynamic Programming Design Paradigm.
2. Compare and analyze the performance of DP algorithms (Time and Space).
3. Extend the solution to not store the whole table (Space Optimization).

Pre-read/Terminologies

1. Priori Analysis and Posteriori Testing.
2. Asymptotic Analysis: Time and Space Complexity in big O, big Theta (Θ), and big Omega (Ω).
3. Algorithm Design Paradigm: Dynamic Programming; LCS: Longest Common Subsequence [2] - Page 84.
4. Recursion.

Resources

Gitlab Repository: https://git.cs.dal.ca/courses/2022-winter/csci-4118-6105/lab2/???? Where ???? is your CSID

Procedure

Note: Undergraduate students are encouraged to attempt additional questions meant for graduate students and get bonus points.

1. Clone the repository: *git clone https://git.cs.dal.ca/courses/2022-winter/csci-4118-6105/lab2/????.git* where *????* is your CSID.
2. Get the latest pull of the master branch: *git pull origin main*
3. Execute all three variations of LCS (Longest Common Subsequence) algorithms for different input sizes and patterns [1].
    a. Refer to the classes *"Driver.java"* and *"LCS.java"*.
    b. Note that the *Driver* class only has an example for *memoization;* make necessary code changes to run the experiment on other variants available in *"LCS.java"*
4. Analyze and compare the time and space (run-time and memory) taken by all variations.

    Extension for Graduate students:

5. Optimize the algorithm to reduce the space complexity to linear and repeat step #4.

Compute the space used and runtime of the program (All Students)

1. Capture the time and space taken by the java program using *java.time* and *runtime* or equivalent in other programming languages.
2. Executing the program:
    a. If you are using an IDE such as IntelliJ (Recommended), build the project and run Driver.main()
    b. Optionally, you can run the java program from the command line: *javac lab2/*.java* (compile) and *java lab2.Driver* (Execute).

Questions

1. Which variant of the LCS algorithm performs the best (space and time)? Explain and mention the actual run-time and memory consumed in your tests.
2. What is the worst-case time and space complexity of LCS recursive solution without memoization, with memoization and the iterative solution? (Methods: recursion, memoization, and iterative in LCS.java) - Use big O notation.

    Extension for Graduate students:

3. Write a program for space-optimized LCS algorithm (Linear Time) and Include the space-optimized variant in Q2.

Submission

Note: For submission - git add, commit and push the answers to the lab2/???? repository and verify the submission in the GitLab web interface.

1. Answer the questions in *"questions_part_1.txt"* (Q1 and Q2 in "questions" section)

    Extension for Graduate students:

2. Add another function "optimized" in the class "LCS" for Longest Common Subsequence taking linear space, and repeat step #2 (4 variations: recursion, memoization, iterative, and optimized).

Grading

Task	3 Points (x1)	2 Points	0 Points
1	Q1 answered correctly.	Q1 answered with partially correct answers.	Incorrect answer or not answered.

Task	2 Points (x1)	1 Point	0 Points
2	Q2 answered correctly without any mistakes.	Q2 answered with partially correct answers.	No evidence of testing.

Task	5 Points (x1)	3 Point	0 Points
3	Space optimized LCS algorithm implemented and tested.	Space optimized LCS algorithm implemented, logically correct (pseudo-code), and/or evidence for testing and explanation.	Incorrect answer or not answered.

References

[1] Wikipedia Contributors, “Longest common subsequence problem,” Wikipedia, Aug. 06, 2019. https://en.wikipedia.org/wiki/Longest_common_subsequence_problem (accessed Feb. 04, 2022).

[2] Matthias Muller-Hannemann and Stefan Schirra. Algorithm engineering: bridging the gap between algorithm theory and practice. Springer-Verlag, Berlin, Heidelberg (accessed Feb. 04, 2022).