The Longest Sorted Bucket Sequence: An Algorithmic Bridge Between LCS and 2D LIS

Awwal Mohammed[1], Caroline S. Selvarajah[2]

[2] Faculty of Arts and Science, Universiti Malaya-Wales, Malaysia
[1]xxxxx@tuta.io; [2]xxxxxx@umwales.edu.my

ABSTRACT

This paper introduces the Longest Sorted Bucket Sequence (LSBS) problem, an algorithmic conceptual framework that extends classical subsequence analysis by incorporating ordered, non-reusable bucket-based constraints. We provide a formal definition of LSBS and demonstrate its fundamental connection to the Longest Common Subsequence (LCS) problem, showing that any LCS instance can be efficiently transformed into an LSBS instance. We then develop two distinct algorithms that solves LSBS: a straightforward $\mathcal{O}(M^2)$ time sparse dynamic programming approach; and a more sophisticated $\mathcal{O}(M \log M)$ time algorithm, where M is the size of the matching list (or the second row of what we call the Common Element Table (CET)). The efficient algorithm is achieved by employing a two-dimensional (2D) sort which in essence embodies the point domination problem, followed by a typical one-dimensional (1D) Longest Increasing Subsequence (LIS) search, a technique analogous to Patience Sorting. We show that this methodology effectively rediscovers the principles of the renowned Hunt-Szymanski algorithm for LCS, offering a simplified and intuitive perspective on its mechanics, while bounding auxiliary space complexity to $\mathcal{O}(|M|)$ in a fashion inspired by but not as strict as Hirschberg's widely acclaimed linear space LCS solution. The total time complexity for solving LCS using this LSBS strategy is $\mathcal{O}(|A| + |B| + M \log M)$ where $|A|$ and $|B|$ are the lengths of the input sequences. Our analysis is supported by implementations and decussated with discussion of trade-offs with other known LCS algorithms, validating the correctness and performance of the proposed framework.

Keywords: Longest Common Subsequence, Longest Increasing Subsequence, Sorting, Dynamic Programming, 2D Point Dominance

I. INTRODUCTION

The study of sequences and subsequences is a widely explored undertaking in computer science, with profound applications in bioinformatics, data compression, text analysis, and beyond. The Longest Common Subsequence (LCS) and Longest Increasing Subsequence (LIS) problems are canonical examples, representing fundamental challenges that have been studied for decades, with the former being notably published by Wagner & Fischer [1] and the latter, Erdős & Szekeres [2]. While elegant solutions for these problems are well-established, the exploration of related variants continues to yield theoretical insights and practical tools.

In this paper, we introduce a conceptual framework called the Longest Sorted Bucket Sequence (LSBS) problem. LSBS involves selecting a sequence of numbers from an ordered collection of "buckets", each of which contain distinctly sorted numbers, subject to the constraint that each chosen number must be greater than the previous (i.e. strictly increasing), and each must originate from a later bucket than the previous. This problem formalizes a structure inherent in many real-world scenarios such as identifying monotonically increasing trends in time-series data grouped by sampling intervals [3], or scheduling tasks that are batched by priority level [4].

The primary contribution of this work is twofold. First, we establish a formal reduction, demonstrating that the classical LCS problem can be transformed into an instance of LSBS. This is achieved by constructing a Common Element Table (CET) data structure, where for each character in the first sequence A, we create a bucket containing the indices of its occurrences in the second sequence B. An LCS of A and B then corresponds directly to an LSBS of this bucket collection.

Second, we explore algorithms to solve the LSBS problem itself, beginning with a baseline dynamic programming solution and then develop an efficient $O(M \log M)$ algorithm, where M is the total number of elements across all buckets. This algorithm lays its foundational momentum on a specialized "bilateral sort" technique followed by a one-dimensional LIS computation, a process that is functionally equivalent to the Patience Sorting algorithm [5]. In doing so, we demonstrate that our LSBS-centric approach to solving LCS provides a new and powerful lens for understanding the Hunt-Szymanski algorithm [6] (one of the most efficient methods for finding LCS) while distilling the complicated aspects using easily understood data structures and algorithmic modelling techniques.

Given the advent and popularity of LCS and LIS in computer science discourse and algorithms literature, we have pragmatically focused on delivering our core algorithms and the processes around them, while deprioritizing verbosity of literature review because these topics have been extensively and rigorously covered by countless high-profile researchers [7, 8, 9, 10]. We instead provide an alternatively effective and compact summary of existing relevant works, while urging readers to refer to aforementioned cited sources for exhaustive coverage.

The rest of the paper is then structured as follows: Section II provides a highly concise and minimalist summary of existing works on LCS and LIS. Section III covers preliminaries on LCS and LIS. Section IV formally defines the LSBS problem and details the reduction from LCS. Section V being the most important part of this paper, presents our algorithmic solutions and analysis for LSBS, and explicitly detailed connection to the Hunt-Szymanski algorithm. Section VI discusses applications and directions for future work, as well as experimental validation and trade-offs, while we conclude in Section VII with a wrap up. A link to the actual implementation hosted on GitHub is provided [11].