An Analysis: Post-Mortem

Arsh Kadakia, Matthew Leung, Ben Natra, Sean Wu

UTEK 2020

Group 7

Our Model

- Algorithm Choice:
    - We used the **Wagner-Fischer Algorithm** to compute the minimum costs for each string conversion.
    - We then used backtracked to determine the process to which the **minimum cost** was arrived.
    - This process was then noted through a separate list.

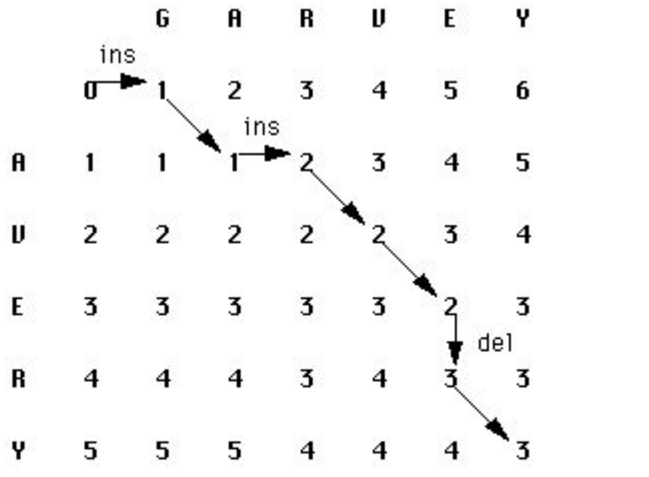
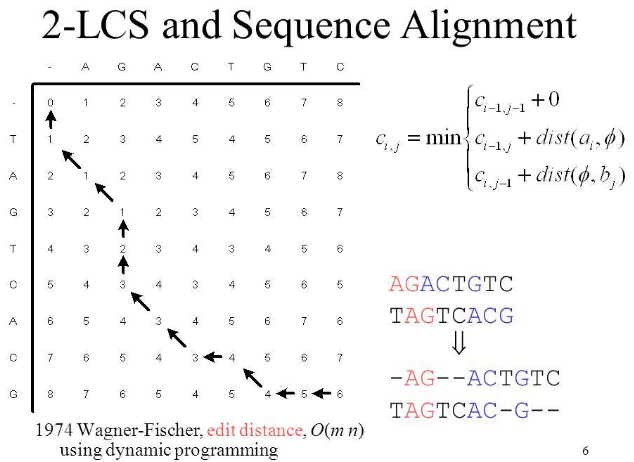


Figure 1. d(i,j) Matrix with Minimal Path Identified

Rationale

- Inspiration came from the minimax algorithm, where **future states** are evaluated to consider which **present path** to take.
- Wagner-Fischer has a runtime of **O(m x n)**, where m and n are the lengths of the input strings
- This **dynamic programming** algorithm is better than some other algorithms that could have worst-case runtime of **O(n^3)**
- Furthermore, it was reliable as it calculated the **minimum edit distance**, which represented the goal of the challenge.



Features

- To run the file, type in command line:
    - `python <script_name>.py <filename>.in`
- Hash table for the names of the books
    - Every book title was assigned an integer
    - Comparisons between integers are faster than comparisons of long strings
    - Hash table format: `[[0, book1], [1, book2], [2, book3]...]`

Steps of the Algorithm

1. Compute the minimum-cost matrix of size (m+1) x (n+1), called `dp`.
2. Backtrack and note down process to get to zero cost.
3. Track all of the potential paths.
4. Determine the optimal path through measuring which path has most consecutive commands.
5. Convert to printable syntax.

The Parallel Problem

- All of the replaces are intended to be grouped together into one command
- Inserts and deletes are done in parallel
  - It does not account for the past deletions or insertions which is why it is incorrect
- For example, when a deletion occurred, all future replace and insert indexes are then modified.
  - The algorithm did not take track of that.

Test Cases and `dp` Matrix

Kitten → Sitting

Replace 0, ‘s’

Replace 4, ‘i’

Insert 6, ‘g’

Sunday → Saturday

Insert 1, 'a'

Insert 2, 't'

Replace 2, 'r'

		S	i	t	t	i	n	g
	0	1	2	3	4	5	6	7
K	1	1	2	3	4	5	6	7
i	2	2	1	2	3	4	5	6
t	3	3	2	1	2	3	4	5
t	4	4	3	2	1	2	3	4
e	5	5	4	3	2	2	3	4
n	6	6	5	4	3	3	2	3

		S	a	t	u	r	d	a	y
	0	1	2	3	4	5	6	7	8
S	1	0	1	2	3	4	5	6	7
u	2	1	1	2	2	3	4	5	6
n	3	2	2	2	3	3	4	5	6
d	4	3	3	3	3	4	3	4	5
a	5	4	3	4	4	4	4	3	4
y	6	5	4	4	5	5	5	4	3

What worked

1. DP Matrix
2. Reading and parsing input file from command line
3. The functions, including the parser, written to write to an output file.
4. The function outputted the correct replace commands.

Possible Improvements

1. Account for the deletion and insertion of books within the given inputs.
    a. The algorithm provided indexes that did not reflect delete actions.
2. We had issues implementing batching of consecutive actions of the same type (ex. Grouping multiple insertions)
3. Preferably, we would have tested the validity of multiple different integrated functions at the same time.

Improvements to our Design Process

1. A more diverse set of testing cases would help us identify potential issues earlier
2. More comments in the code also would have helped readability and make the code easier to improve over time

Thank you for the opportunity!