

# Introduction to String Transformation

This presentation will explore the fascinating concept of string transformation, where we can manipulate a string by removing a suffix and appending it to the beginning. We'll dive into the problem statement, the approach, and various techniques to handle complex scenarios.

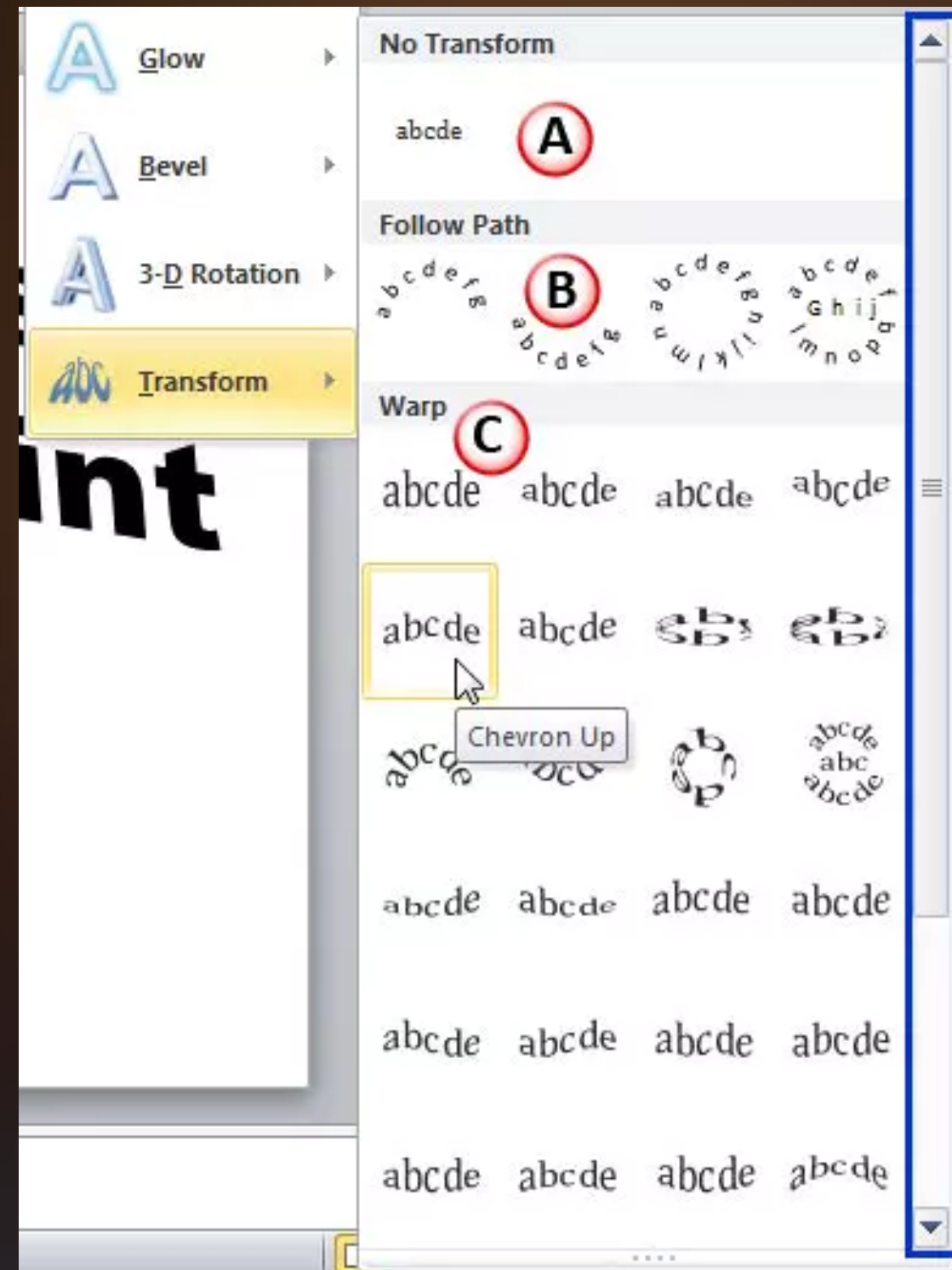


BY

Mano Karthik Reddy .V

192225077

AI&ML



# Problem Statement

1

## Given Strings

Two strings, s and t, of equal length n.

2

## Transformation Operation

Remove a suffix of s with length l, where  $0 < l < n$ , and append it to the start of s.


3

## Goal

Find the number of ways to transform s into t in exactly k operations.

## 2. THE PROBLEM

*Our teams are overrun with emergencies and urgent requests.  
This means we have no time to innovate, or to get the best from staff.  
If this continues, our competitors will push us out of the market.*

RAG	Summary	Business Area Status
	<div>1. Legacy overhead is swamping our teams.</div> <div>2. There is no space for feature experiments or new ideas.</div> <div>3. Our competitors are moving ahead.</div> <div>4. Our best staff are leaving.</div>	<div>WIDGET TEAM: Maxed out.</div> <div>PR TEAM: Need new feature.</div> <div>PLATFORM: Need rebuild.</div> <div>INNOVATION: no available time</div>

# Approach Overview

## Generating Transformations

Systematically generate all possible transformations by removing and appending suffixes.

## Tracking Steps

Keep track of the number of operations required to transform  $s$  into  $t$ .

## Optimizing Complexity

Implement efficient techniques to handle large inputs and optimize time complexity.

# Generating Possible Transformations

1

## Identify Suffixes

Iterate through all possible suffix lengths (1 to n-1) and remove each suffix.

2

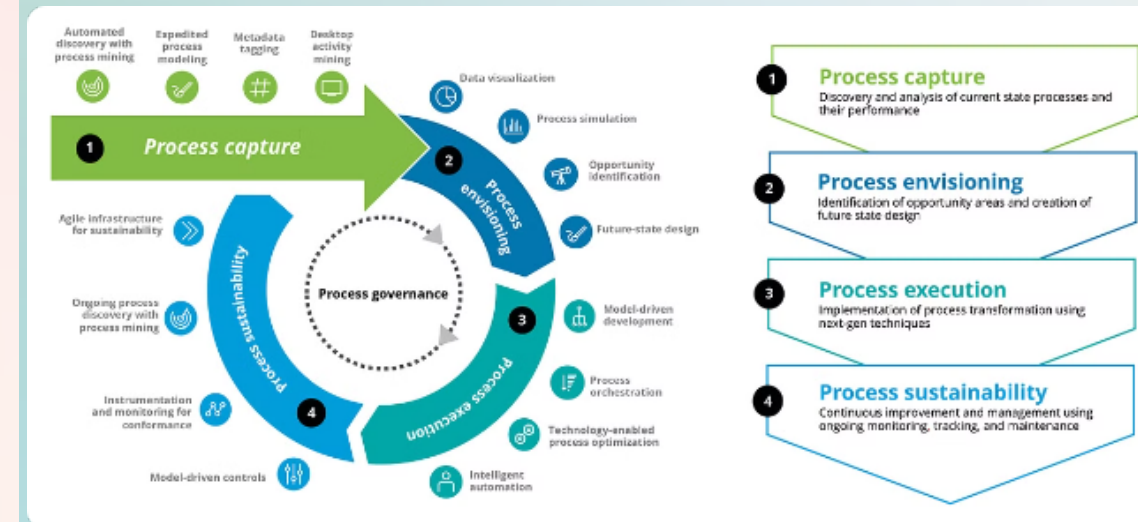
## Append to Beginning

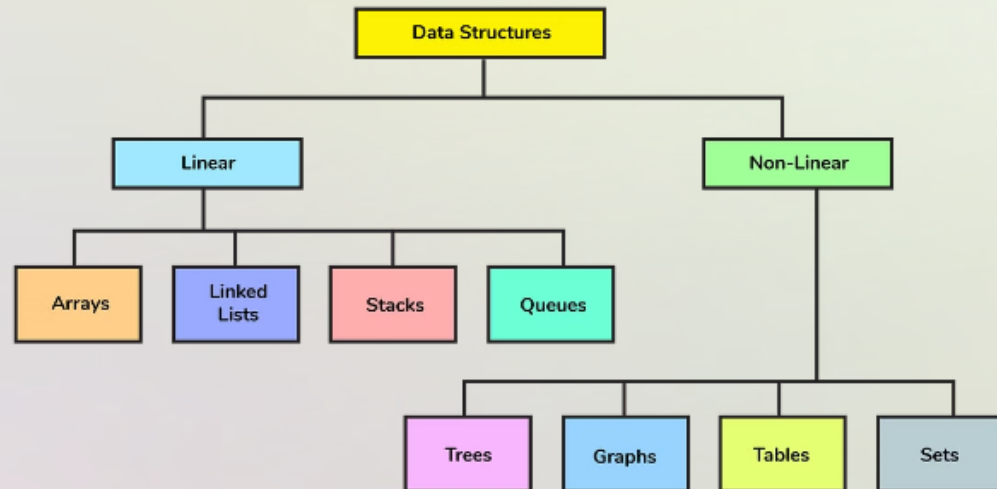
Append the removed suffix to the start of the string.

3

## Track Transformation

Record each transformation and the number of operations required.





# Tracking Transformation Steps

## Memoization

Use a hash table or a dynamic programming approach to store and reuse previously computed transformations.

## Backtracking

Explore all possible transformation paths and keep track of the number of operations performed.

## Iterative Approach

Implement an iterative solution to efficiently traverse the transformation space.

## Pruning

Identify and avoid redundant or unnecessary transformations to optimize the search process.

# Handling Large Inputs

## Memory Management

Implement techniques to efficiently store and access the transformation data, especially for large inputs.

## Parallelization

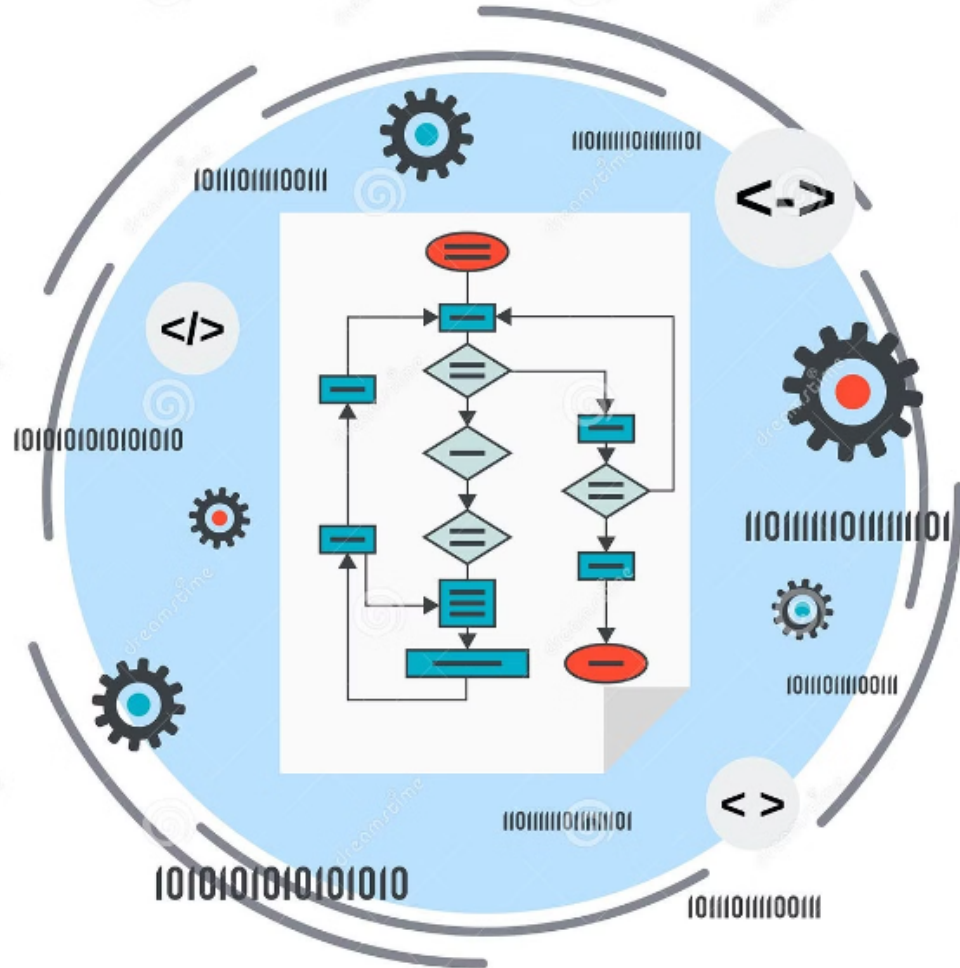
Explore opportunities for parallel processing to speed up the transformation search on multi-core systems.

## Approximation Algorithms

Develop heuristic-based approaches to provide approximate solutions for large inputs when an exact solution is infeasible.



# Optimizing Time Complexity



[dreamstime.com](http://dreamstime.com)

ID 54561131 © Vadim Ermak



## Complexity Analysis

Thoroughly analyze the time and space complexity of the proposed algorithms and identify areas for optimization.



# Dynamic Programming

Leverage dynamic programming techniques to reuse previously computed results and avoid redundant work.



## Pruning Techniques

Develop efficient pruning strategies to eliminate unnecessary transformations and reduce the search space.



## Parallel Processing

Explore opportunities for parallelization to distribute the transformation search across multiple processors or threads.

# Edge Cases and Considerations

1

## Empty Strings

Handle the case where  $s$  or  $t$  is an empty string.

2

## Identical Strings

Consider the scenario where  $s$  and  $t$  are initially the same.

3

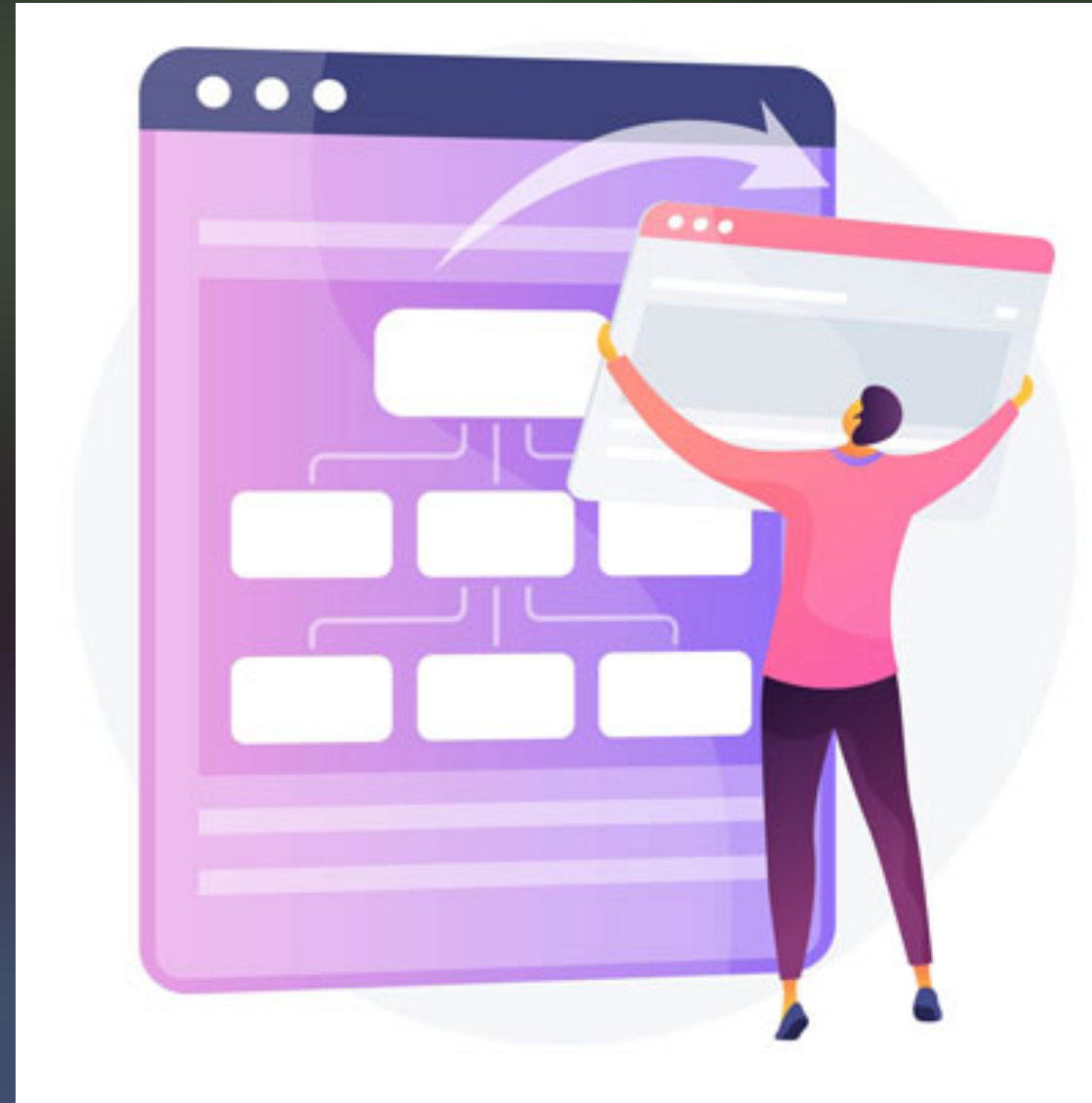
## Negative $k$

Gracefully handle the case when  $k$  is negative or exceeds the maximum number of possible transformations.

4

## Precision Handling

Ensure accurate counting of the number of transformations, especially for large values.







# Implementing the Solution

## Algorithm

### Suffix Removal and Appending

## Pseudocode

1. Iterate through all possible suffix lengths (1 to  $n-1$ )
2. Remove the suffix from  $s$
3. Append the removed suffix to the start of  $s$
4. Record the transformation and the number of operations

### Memoization and Backtracking

1. Use a hash table to store previously computed transformations
2. Explore all possible transformation paths
3. Keep track of the number of operations performed
4. Prune unnecessary transformations to optimize the search

# Conclusion and Key Takeaways

1

## Systematic Approach

Develop a structured approach to generate and track string transformations.

2

## Optimization Techniques

Leverage memoization, pruning, and parallel processing to improve efficiency.

3

## Edge Case Handling

Carefully consider and address edge cases to ensure robust and reliable solutions.

4

## Continuous Improvement

Analyze and refine the algorithms to achieve better time and space complexity.

