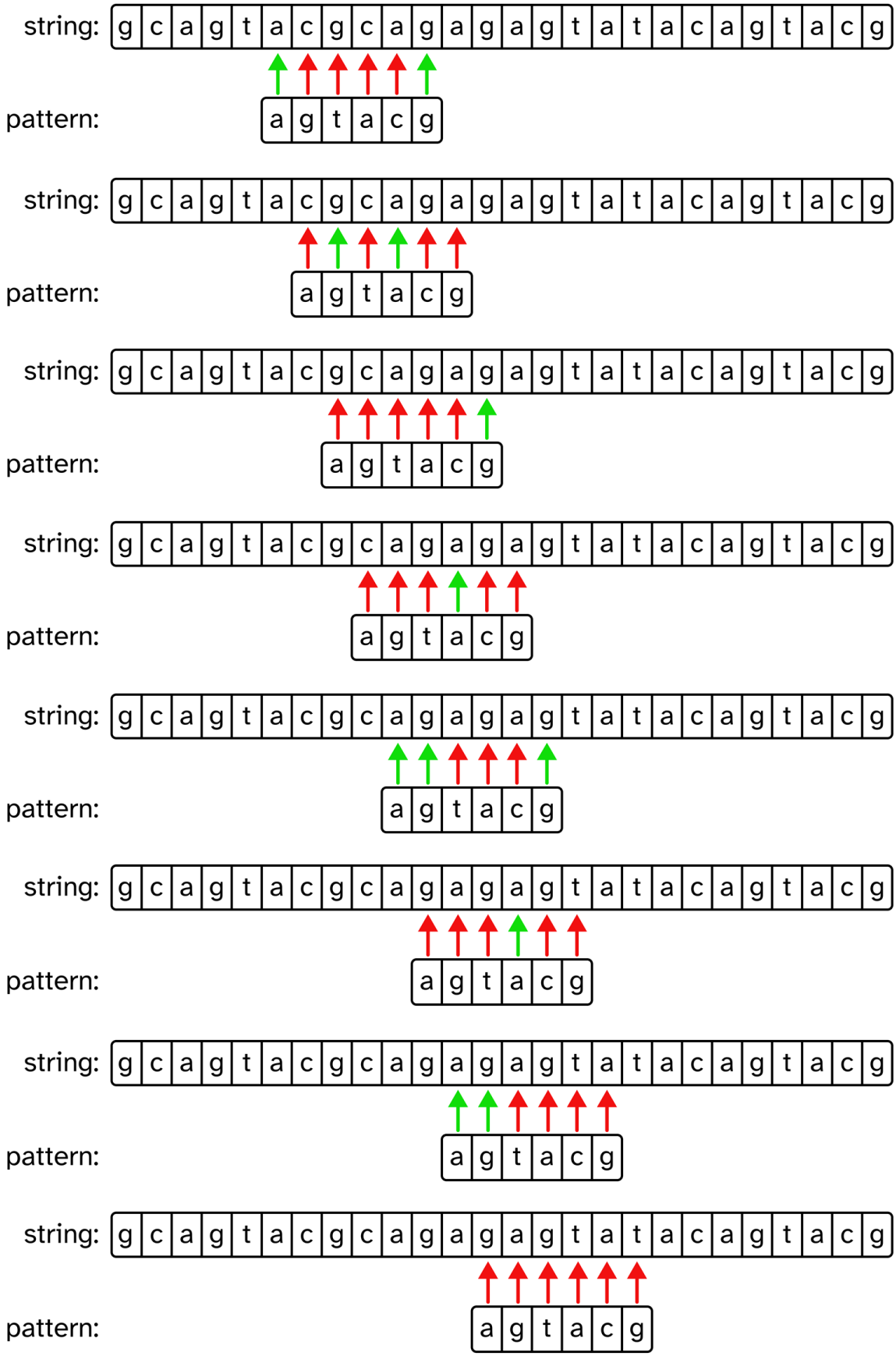
Ryan Christopher
MET CS566
Assignment 5

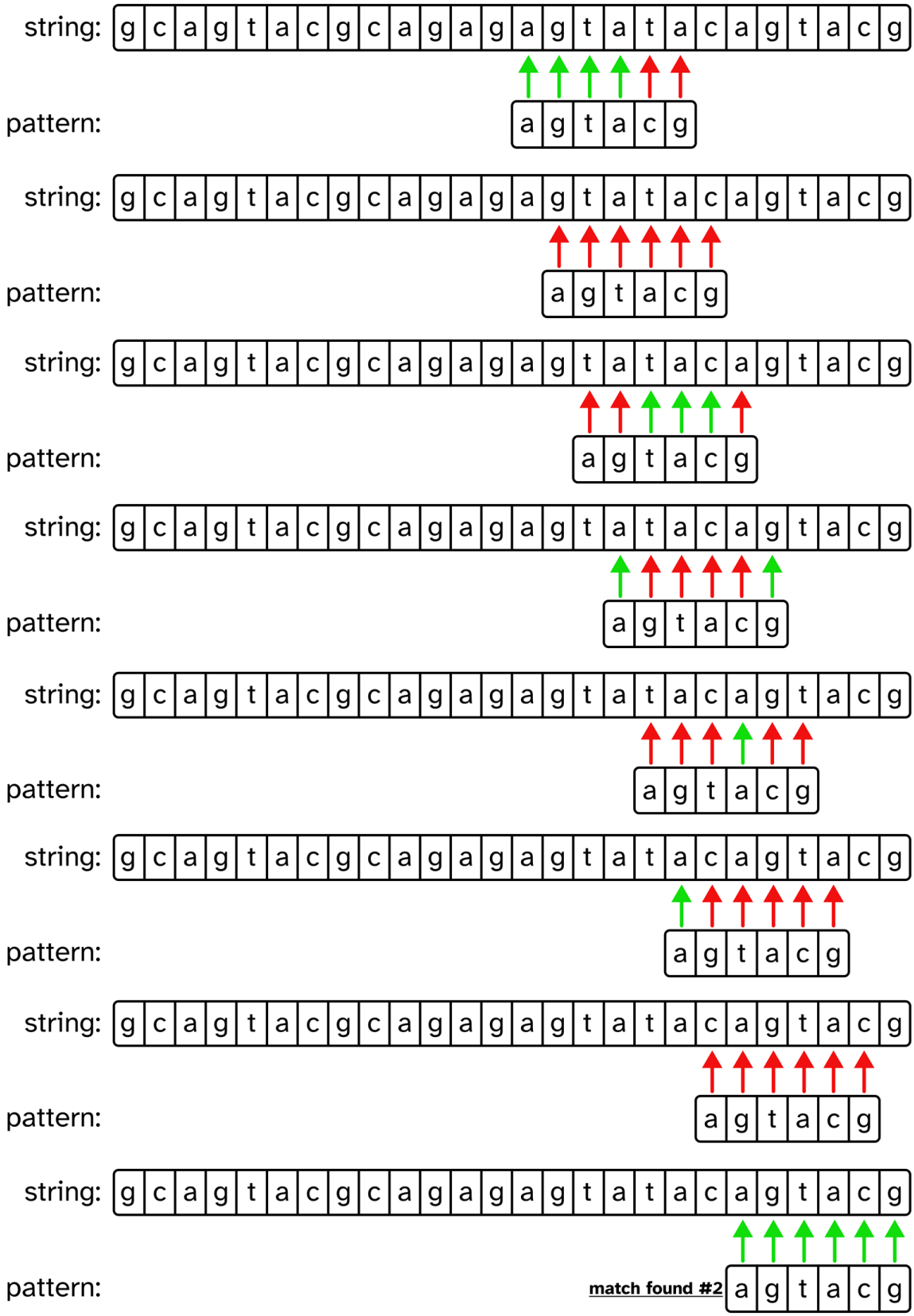
1. Evaluate performance characteristics of two string matching algorithms (Brutal Force algorithm and Knuth-Morris-Pratt algorithm) for the following case:

Pattern: agtacg
String: gcagtacgcagagagtatacagtacg

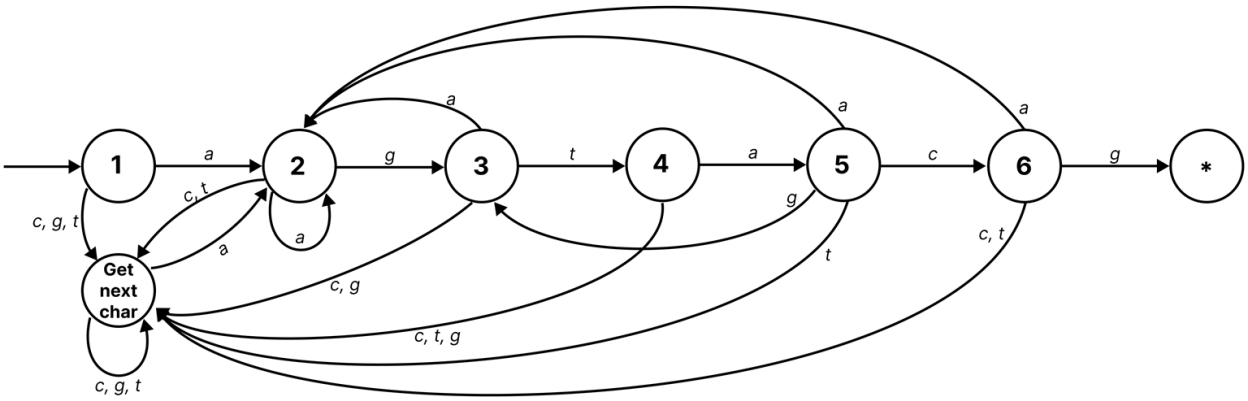
Compare the performance of these two algorithms in terms of **preprocessing time** and **matching time**.

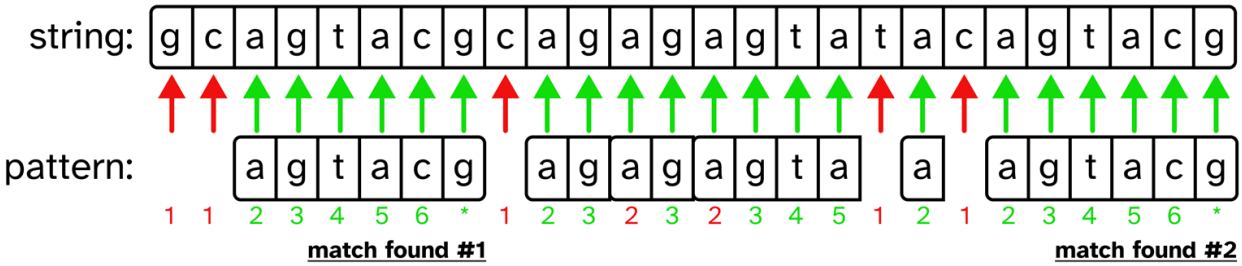
Brute Force:


string: g c a g t a c g c a g a g a g t a t a c a g t a c g
pattern: a g t a c g
string: g c a g t a c g c a g a g a g t a t a c a g t a c g
pattern: a g t a c g
string: g c a g t a c g c a g a g a g t a t a c a g t a c g
pattern: a g t a c g
string: g c a g t a c g c a g a g a g t a t a c a g t a c g
pattern: a g t a c g
string: g c a g t a c g c a g a g a g t a t a c a g t a c g
pattern: a g t a c g
string: g c a g t a c g c a g a g a g t a t a c a g t a c g
pattern: a g t a c g
string: g c a g t a c g c a g a g a g t a t a c a g t a c g
pattern: a g t a c g
string: g c a g t a c g c a g a g a g t a t a c a g t a c g
pattern: a g t a c g


string: g c a g t a c g c a g a g a g t a t a c a g t a c g
pattern: a g t a c g
string: g c a g t a c g c a g a g a g t a t a c a g t a c g
pattern: a g t a c g
string: g c a g t a c g c a g a g a g t a t a c a g t a c g
pattern: a g t a c g
string: g c a g t a c g c a g a g a g t a t a c a g t a c g
pattern: a g t a c g
string: g c a g t a c g c a g a g a g t a t a c a g t a c g
pattern: a g t a c g
string: g c a g t a c g c a g a g a g t a t a c a g t a c g
pattern: a g t a c g
string: g c a g t a c g c a g a g a g t a t a c a g t a c g
pattern: a g t a c g
match found #2
string: g c a g t a c g c a g a g a g t a t a c a g t a c g
pattern: a g t a c g

Knuth-Morris-Pratt:





Preprocessing Time – For this problem, the brute force algorithm requires O(1) processing time as the algorithm does not perform any operations in the preprocessing phase. Knuth-Morris-Pratt, however, requires O(m) time for preprocessing, where m is the length of the pattern string that is being searched for.

Matching Time – Brute force requires an average of O(n + m) time for matching, where n is the input string and m is the pattern string, however in this case it is closer to O(nm) as there were many sub-strings in n that began with the same characters as the pattern string. Knuth-Morris-Pratt requires only O(n) for matching as the string is processed through a DFA and only requires iterating through n once.

2. Design and implement (i.e., write a program) an algorithm based on **text matching** that adds two polynomials such as:

$$(x^{**}3 + 5x^{**}2 - 3x + 3) + (4x^{**}5 - 2x^{**}2 + 1)$$

Place the theta value of your algorithm as the first comment in the code. Submit several challenging test cases.
Hint: Input consists of a text string.

main.rs contains the function “add_polynomials” which uses text matching to add two polynomials together. Below is the output of the test cases:

```
========== Test Case 1 ==========
(x**3 + 5x**2  - 3x + 3) + (4x**5 - 2x**2 + 1)
Expected: 4x**5 + x**3 + 3x**2 - 3x + 4
Result:   4x**5 + x**3 + 3x**2 - 3x + 4

========== Test Case 2 ==========
(-14x**2 - 8x + 1) + (15x**2 - 8x -1)
Expected: x**2 - 16x
Result:   x**2 - 16x

========== Test Case 3 ==========
(2 + 50x**9 - 13x**3 - 13x**4) + (37x**5 + 5 - 2x**4)
Expected: 50x**9 + 37x**5 - 15x**4 - 13x**3 + 7
Result:   50x**9 + 37x**5 - 15x**4 - 13x**3 + 7

========== Test Case 4 ==========
(x + 2x**2 + 3x**3 + 4x**4 + 5x**5) + (-x - 2x**2 - 3x**3 - 4x**4 - 5x**5)
Expected: 0
Result:   0

========== Test Case 5 ==========
(82x**20 + 523x**17 - 39x**5) + (195x**10 - 2x**8 - 620)
Expected: 82x**20 + 523x**17 + 195x**10 - 2x**8 - 39x**5 - 620
Result:   82x**20 + 523x**17 + 195x**10 - 2x**8 - 39x**5 - 620

========== Test Case 6 ==========
(-10x**200 - 125x**189 - x**99 - 55x - 20) + (-x**290 - 62x**53 - x**20 - x - 100)
Expected: -x**290 - 10x**200 - 125x**189 - x**99 - 62x**53 - x**20 - 56x - 120
Result:   -x**290 - 10x**200 - 125x**189 - x**99 - 62x**53 - x**20 - 56x - 120
```

3. What is your **informed** opinion on the NP problems? Does a deterministic NP polynomial time algorithm exist or not?

NP problems represent a substantial hurdle within computer science, technology, and math. The goal of determining if every problem that can be efficiently solved and verified irrespective of the input size poses as a great unknown that is difficult to prove and disprove. While over time a number of problems that were thought to be NP had a solution discovered and were proven to be P, I do not think that a deterministic NP polynomial time algorithm exists.

Specifically, the subclass of NP problems called NP-complete serve as the barrier for proving/disproving P = NP, and have been grouped such that if one NP-complete class problem is proven to be P, it would therefore mean all NP problems can be P. The first NP-complete problem was introduced in 1971 with Stephen Cook's SAT Problem, then with later additions of the Knapsack Problem, and the Traveling Salesman Problem. In the 50+ years since the problems have been introduced, there have been no solutions that solve them in polynomial time. Due to the nature of the NP-complete questions and how they require exponentially greater computations as their inputs scale, my thought is that P != NP.

4. Give an example of non-linearities in neural networks used for machine learning. Why is it necessary to have it in networks? Can feature extraction improve the performance of fully connected neural networks used for handwritten digit recognition?

The sigmoid function is a commonly used non-linearity used in neural networks for machine learning. These non-linearities are necessary in networks as they allow for multiple layers to be introduced in order for more complex operations to occur. Without them neural networks would only behave linearly, and would not be capable of scaling to larger complexities. In particular, the sigmoid function is used for narrowing a result down to a value between 0 and 1 which can be used to determine if a node will be activated or not. This allows for large amounts of data to be processed and fitted to a range that allows a network to add multiple layers of complexity that nodes can activate efficiently.

Feature extraction can improve the performance of a fully connected neural network that is used for handwritten digit recognition. By using feature extraction, the images fed to the network can have their dimensionality reduced which increases efficiency and aids in detecting edges that can be used to identify which numbers are written.